CRREL REPORT 92-22

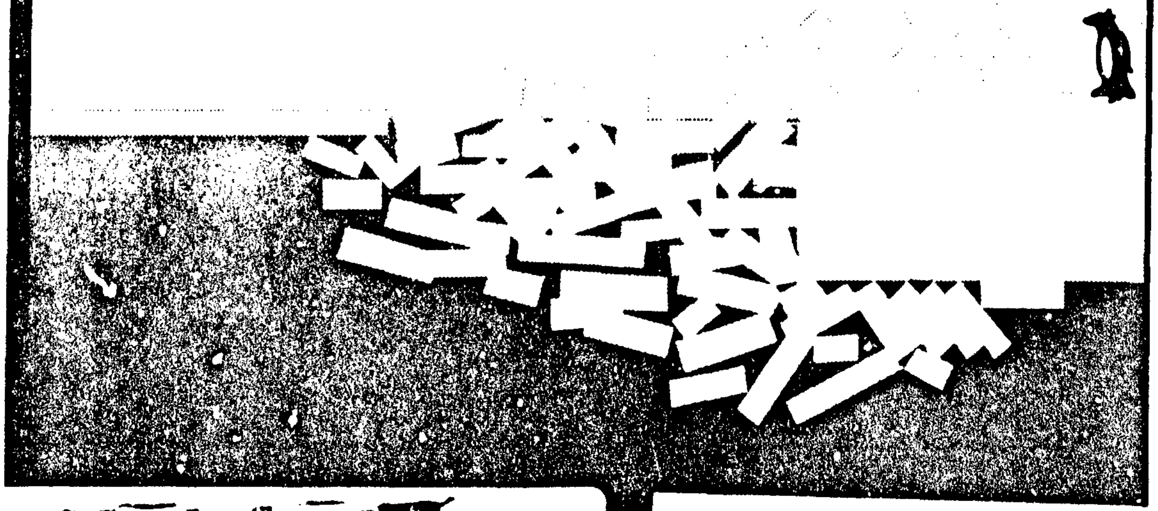
AD-A262 556

CRREL

Numerical Simulation of Systems of Multitudinous Polygonal Blocks

Mark A. Hopkins

December 1992



Abstract
In this work a method is developed for the simulation of two-dimensional systems of discrete convex polygonal particles. The particles themselves are rigid while the interparticle contacts are deformable. The contacts are termed deformable because areas of overlap that are found between adjacent particles are interpreted as surface deformations. The contact forces between particles are calculated from an elastic-viscous-plastic model based on the area of overlap and the rate of change of the area of overlap. The magnitude of the contact force is limited by the strength of the material. If the contact force reaches this limit, further deformation is non-recoverable.

Cover: Simulated ridging of a sheet of unbroken ice covering a lead (with virtual penguin).

CRREL Report 92-22

U.S. Army Corps of Engineers
Cold Regions Research & Engineering Laboratory

Numerical Simulation of Systems of Multitudinous Polygonal Blocks

Mark A. Hopkins

December 1992

Prepa.ed for
OFFICE OF NAVAL RESEARCH

PREFACE

This report was prepared by Mark A. Hopkins, Research Physical Scientist, Ice Engineering Research Branch, Experimental Engineering Division, U.S. Army Cold Regions Research and Engineering Laboratory. This work was begun at the Thayer School of Engineering, Dartmouth College, Hanover, New Hampshire, with the support of the Office of Naval Research University Research Initiative N00014-86-K-069, and finished at CRREL.

Technical review of this work was provided by Michael G. Ferrick and Dr. James Lever, both of CRREL.

CONTENTS

ILLUSTRATIONS

Numerical Simulation of Systems of Multitudinous Polygonal Blocks

MARK A. HOPKINS

1. INTRODUCTION

Macroscopic systems of discrete particles in motion, given the general name of granular flows, occur in many industrial and geophysical processes. Examples of such flows are found in powder technology, in grain handling, in slurry transport by pipeline, in avalanches, and in the transport of sediment and ice by river and ocean.

The Arctic Basin, covered with a permanent ice pack, is the grandest of granular flows. The particles in this system are ice floes with diameters of 1–10 km. The ice pack circles the North Pole in a clockwise path to merge with the Trans-Polar Drift and leaves the Arctic through the Fram Strait between Greenland and Spitsbergen. Convergence of the ice pack produces local failure or pressure ridging of the ice-covered leads separating the floes. The growing pressure ridges are slowly deforming granular systems with block-like particles of about 1 m in length.

A computer simulation was applied to granular flows in 1980 by Otis Walton at Lawrence Livermore. Walton's work was based on the discrete element quasi-static simulations of rock assemblies pioneered by Peter Cundall (1971, 1974, 1980, Cundall et al. 1978). Walton (1980) describes the conceptual basis for a two-dimensional Newtonian simulation of systems of polygonal particles, which he calls the Discrete Interacting Block System (DIBS). In the DIBS code, the particles themselves are rigid. Internal deformation of the particles is not modeled and therefore, their shape does not change in response to applied forces. Interparticle contacts, the areas of overlap between adjacent polygons that change from time step to time step, are interpreted as surface deformations.

In the present work a similar method is developed for the simulation of two-dimensional systems of discrete, convex, polygonal particles. While the general development follows Walton, several improvements are introduced. The most important is the use of an elastic–viscous–plastic contact force model based on the area of overlap and the rate of change of the area of overlap. The magnitude of the contact force is limited by the compressive strength of the material. If the contact force reaches this limit, further deformation is non-recoverable. A Mohr–Coulomb frictional force acts tangentially at the point of contact. This work describes several features useful for sea ice and river ice modeling, including flexural failure of slender blocks and a dynamic beam simulation for modeling failure in intact ice sheets. One further improvement over previous methods is the clarity and relative simplicity of the computer code itself.

The goal of a dynamic simulation of a system of discrete particles is to reproduce the significant behavior of the real system. The simulation may be used to perform a numerical experiment. A numerical experiment based on a particle simulation has several advantages over a corresponding physical experiment. The first advantage is the ease of obtaining statistical averages of variables descriptive of the system's behavior. This is because the particle positions and velocities and the forces exist explicitly in the simulation and thus the various moments, stresses and energies may be easily found. The second advantage is the ability to selectively manipulate the physics of a problem in ways that are impossible in a physical experiment. Examples would be the elimination of gravity, the elimination of temperature effects in ice experiments, and the use of periodic boundaries to remove the inhomogeneities present in a material near physical boundaries. The third advantage is that there are no scale limitations in numerical experiments. Large problems like ice ridging or river ice jamming need not be scaled down as they must be in a laboratory. The shortcomings of numerical experiments conducted with dynamic simulations result from the crudeness of the physical models of the interactions between individual blocks and between blocks and the surrounding fluid and the assumed homogeneity of the material.

2. CONSTRUCTION OF THE SIMULATION

In general terms, a particle simulation is a computer program that models the Newtonian dynamics of a large system of discrete particles. The simulation stores the shape and the instantaneous position and velocity of each particle. The contact and body forces on each particle are calculated at each time step and the particles are moved to new locations with new velocities that depend on the forces.

The simulation of systems of asymmetrical polygonal particles is more complicated than the simulation of systems of symmetrical particles, such as disks or spheres. The complexity of the simulation is a consequence of the complexity of the relationships among the particles and the difficulty in defining them. The relationships change constantly as the system deforms time step by time step. The algorithm must individually define and assess the relationships among the many points and sides of the particles to discover those that are in contact. Particles are defined to be in contact when their areas overlap.

After the algorithm has discovered the particles in contact, it must calculate the forces between them. In this work a deformable contact model is used. The contact model is called deformable because the overlaps between neighboring particles, which change from time step to time step, are interpreted as deformations of the surfaces in contact. The deformations are treated as springs and dashpots in compression that generate restoring or repulsive forces between the particles. The interparticle force has an elastic component proportional to the area of overlap and a viscous component proportional to the rate of change of the area of overlap. Mohr–Coulomb frictional forces act tangentially at the point of contact. Both sliding contacts and static contacts are modeled. The effect of the forces on the motions of the particles is found by integrating the equations of motion for each particle at each time step.

The six main components of a particle simulation are:

1. Storage of the instantaneous position and velocity of each particle.
2. Discovery of the particles that are in contact with other particles at a given time.
3. Calculation of the force between those particles.
4. Integration of the equations of motion for each particle at each time step.
5. Calculation of statistical averages of variables of interest.
6. Construction of a visual record of the system's motion.

2.1 Domain of the simulation

The domain of the simulation is the area that the particles inhabit. The boundaries of the domain may be open or closed. Open boundaries typically depend on gravity to confine the particles. A closed boundary may be of any arbitrary shape defined by a sufficient number of closely spaced particles. A closed boundary may be either fixed or movable. The motions of movable boundaries may be either stress or strain controlled. Alternatively, periodic boundaries may be used to model an unconfined domain by causing the particles that leave the domain at one boundary to reenter at the opposite boundary. The equations that control the periodicity are

$$\text{if } x_i > L_x \text{ then } x_i = x_i - L_x \tag{1a}$$

$$\text{if } x_i < 0.0 \text{ then } x_i = x_i + L_x \tag{1b}$$

where the subscript i denotes a particle i in a domain of length L_x. The periodic boundary condition creates the topological equivalent of a cylindrical domain formed by joining two opposing boundaries. Either the x or y boundaries or both may be periodic.

2.2 System configuration

The simulation algorithm must keep a record of the configuration of the system at each time step. The system configuration consists of the position, orientation, velocity and shape of each of the N particles (henceforth polygons). The position of each polygon is stored in the arrays $x(N)$ and $y(N)$. The orientation of each polygon is stored in the array $\theta(N)$. The x and y components of the translational velocities are stored in the arrays $u(N)$ and $v(N)$. The rotational velocities are stored in the array $\omega(N)$. Each of the polygons has Nv vertices stored in an array $Nv(N)$. The shape of each polygon is defined by the length of the rays, from the center of mass of the polygon to each vertex, stored in the array $r(N,Nv_{\max})$ and the angles, measured between a reference point and each ray, stored in the array $\alpha(N,Nv_{\max})$. The mass and polar moment of each polygon are stored in the arrays $mass(N)$ and $Jp(N)$. The system configuration is saved at the end of a given run for restart.

In addition, the following information is carried by the simulation algorithm, but is not part of the system configuration saved in the restart files. The components of the vectors defining the location of the vertices of each polygon relative to its center of mass are carried in the arrays $rx(N,Nv_{\max})$ and $ry(N,Nv_{\max})$. The cosine and sine of the shape-defining angles $\alpha(N,Nv_{\max})$ are carried for the efficient calculation of $rx(N,Nv_{\max})$ and $ry(N,Nv_{\max})$ at each time step.

2.3 Search process

The search procedure performs the task of finding neighboring polygons. The simplest method of going about a search is by choosing one polygon from the flow and then searching amongst all of the remaining polygons for those that are in contact with the chosen polygon. This process is then repeated one by one for the remaining polygons. However, in a system containing N polygons, $N^2/2$ individual searches would be required.

One way of shortening the search process is to limit the search about the chosen polygon to its immediate neighborhood. A neighborhood about a polygon is defined by a uniform square grid superimposed on the domain. The width of the grid cells L_c must be at least as large as the largest dimension of the largest polygon. A three-dimensional array $grid(i,j,k)$ is defined such that the i and j dimensions of the array correspond to the two-dimensional grid, while the third dimension stores the indices of the polygons with centers in cell i,j.

The i and j location in the grid of the center of mass of polygon k are given by

$$i = 1 + \text{int}[x(k)/L_c] \tag{2a}$$

$$j = 1 + \text{int}[y(k)/L_c]. \tag{2b}$$

The presence of polygon k in cell i,j is recorded in the grid array

$$grid[i,j,ngrid(i,j)] = k \tag{3}$$

where $ngrid(i,j)$ is the current number of polygons with centers in cell i,j. The column, row address of polygon k is stored in a pair of one-dimensional arrays

$$invx(k) = i \tag{4a}$$

$$invy(k) = j\,. \tag{4b}$$

These arrays provide a means of finding the grid location of a given polygon and searching its immediate surroundings for neighbors or pending contacts. Figure 1 shows a polygon in the center of a block of grid cells that is part of the grid overlying the flow domain. This block of nine cells is the search neighborhood about the polygon. A second polygon is shown near the perimeter of the search neighborhood. The (i,j) location of the central or home polygon in the grid is given by eq 4a and b. The number of the neighbor or near polygon at $(i + 1,j + 1)$ in the grid is given by eq 3.

The search strategy discussed above is implemented in the following subroutines GRIDFILL and SEARCH. The subroutine GRIDFILL reinitializes the grid, puts each polygon into its proper grid location, and fills the inverse addresses, arrays *invx* and *invy*.

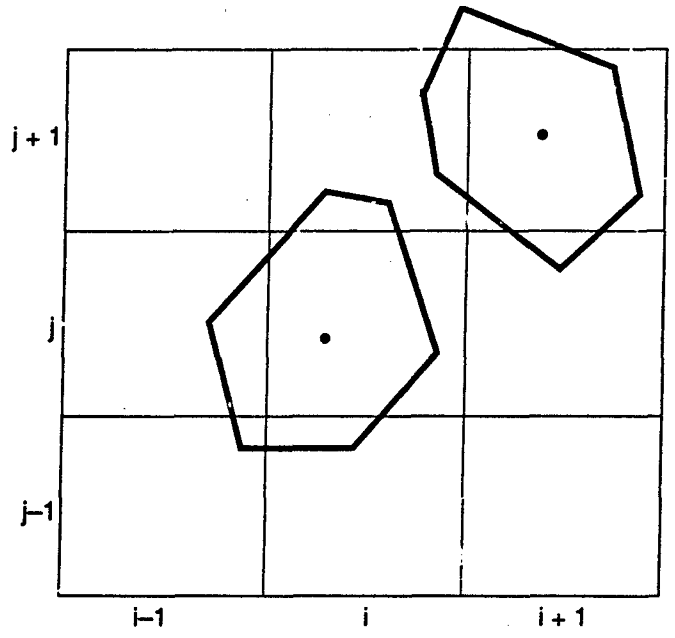


Figure 1. Search neighborhood surrounding a polygon.

2.3.1 Subroutine GRIDFILL

```
Do i = 1,dimi
   Do j = 1,dimj
      ngrid(i,j) = 0
   End Do
End Do

Do i = 1,N
   gi = 1+int(x(i)/Lc)
   gj = 1+int(y(i)/Lc)

   ngrid(gi,gj) = ngrid(gi,gj)+1
   grid(gi,gj,ngrid(gi,gj)) = i

   invx(i) = gi
   invy(i) = gj
End do

End
```

In the SEARCH subroutine, a search is conducted around each polygon in the flow field. The number of the central or home polygon is *nh*. The inverse arrays *invx* and *invy* are used to find the *i,j* position of the home polygon. The search extends over the central grid cell and the eight surrounding cells about *nh*. The number of a neighboring or near polygon in the grid is *nn*. Count is the number of the current pair of polygons in the list of pairs of near neighbors being compiled. The subroutine CONTACT, discussed below, discards neighbors separated by a distance greater than ε.

2.3.2 Subroutine SEARCH

```
Do nh = 1,N
   Do i = invx(nh)-1, invx(nh)+1
      Do j = invy(nh)-1, invy(nh)+1

         Do k = 1,ngrid(i,j)

            nn = grid(i,j,k)

            If (nn .lt. nh) then

               neighbor = .false.
               Call Contact(nn,nh,neighbor)

               If (neighbor) then

                  count = count+1

                  home(count) = nh
                  near(count) = nn
                  nnear(nh) = nnear(nh)+1
            End If
          End
        End I
      End Do
    End Do
  End Do

  End
```

Because the distance traveled by particles between time steps may be extremely small, it is efficient to discard neighboring pairs separated by a distance greater than ε. The chosen value of ε depends on the average distance traveled by particles between calls to the SEARCH subroutine. A brute force algorithm is used to find the smallest separation between pairs of polygons. Every vertex on one polygon is tested against every side of the other polygon until a vertex is found that is closer than ε. The process is performed by the CONTACT subroutine, called from the SEARCH subroutine.

2.3.3 Subroutine CONTACT(nh,nn,neighbor)

```
C Make side vectors for NEAR polygon:

  Do j = 1,Nv(nn)

    jp = 1+mod(j,Nv(nn))

    ssx(j) = (rx(nn,jp)+rx(nn,j))/length(nn,j)
    ssy(j) = (ry(nn,jp)+ry(nn,j))/length(nn,j)

  End Do

C Find vertex of HOME polygon closest to side of NEAR polygon:

  pt = 1

  Do While(.not. neighbor .and. pt .le. Nv(nh))

    dx = x(nh)+rx(nh,pt)-x(nn)
    dy = y(nh)+ry(nh,pt)-y(nn)

    maxsep = 50.0

    Do j = 1,Nv(nn)

      xprod = ssx(j)*(dy-ry(nn,j))-ssy(j)*(dx-rx(nn,j))

      if (xprod .lt. maxsep) maxsep = xprod

    End Do

    if (maxsep =.ge. -epsilon) neighbor = .true.

    pt = pt+1

  End Do
```

```
If (.not. neighbor) then
    {Repeat process - exchanging polygons}
End If
End
```

There is a discontinuity in the numbering of vertices encountered when moving around the polygon in the counter-clockwise direction. If, when k equals $Nv(i)$ (the number of vertices on polygon i), k is incremented, then k would equal $Nv(i)+1$ rather than 1. The CONTACT subroutine uses the mod function to cross the discontinuity. Thus, incrementing the vertex counter k by 1 is done using the equation

$$k = 1 + \mathrm{mod}[k, Nv(i)]. \tag{5}$$

The corresponding equation for decrementing k by 1 is

$$k = 1 + \mathrm{mod}[k-2+Nv(i), Nv(i)]. \tag{6}$$

In summary, to begin the search process, the entire system of polygons is sorted into a grid. By use of the grid, the immediate neighborhood of each polygon is examined to discover pairs of neighboring polygons. The spatial relationship between each pair of polygons is studied to determine the minimum separation distance between the polygons. If this distance is less than some small value ε, then the numbers of the two polygons are stored in the list of near neighbors or potential contacts. The value of ε chosen depends on the average distance traveled by particles between calls to the SEARCH subroutine.

Because the simulation time step is short, the relationship between neighboring pairs of polygons changes little from one time step to the next. Therefore, it is possible to drastically increase the efficiency of the simulation by calling the SEARCH subroutine at intervals of many time steps. The length of the interval depends on the rate of deformation of the system. In practice, as the interval is increased, a point of diminishing returns is reached. This depends on the relative time required to perform the search versus the time required to calculate contact forces.

2.4 Interparticle forces

The calculation of the inter-particle forces begins with the list of pairs of neighboring polygons compiled in the search process described above. A pair of polygons is defined to be in contact when their areas intersect. The intersection between the two polygons, which changes from time step to time step, is interpreted as deformation of the surfaces in contact. The deformation is treated as a spring and dashpot in compression, which generates a repulsive force between the polygons.

The important features of the force calculation strategy are to:

1. Discover whether or not a given pair of polygons intersect.
2. Calculate the area of intersection and the location of the centroid of the area of intersection. The centroid of the area of intersection defines the point of action of the contact force between the pair of polygons.
3. Define a line of contact between the two polygons and a local coordinate frame with axes normal (**n**) and tangential (**t**) to the line of contact.
4. Compute the normal and tangential forces between the polygons and their respective moments in the local (**n**,**t**) coordinate frame.
5. Convert the forces and moments to the global (**x**,**y**) coordinate frame.

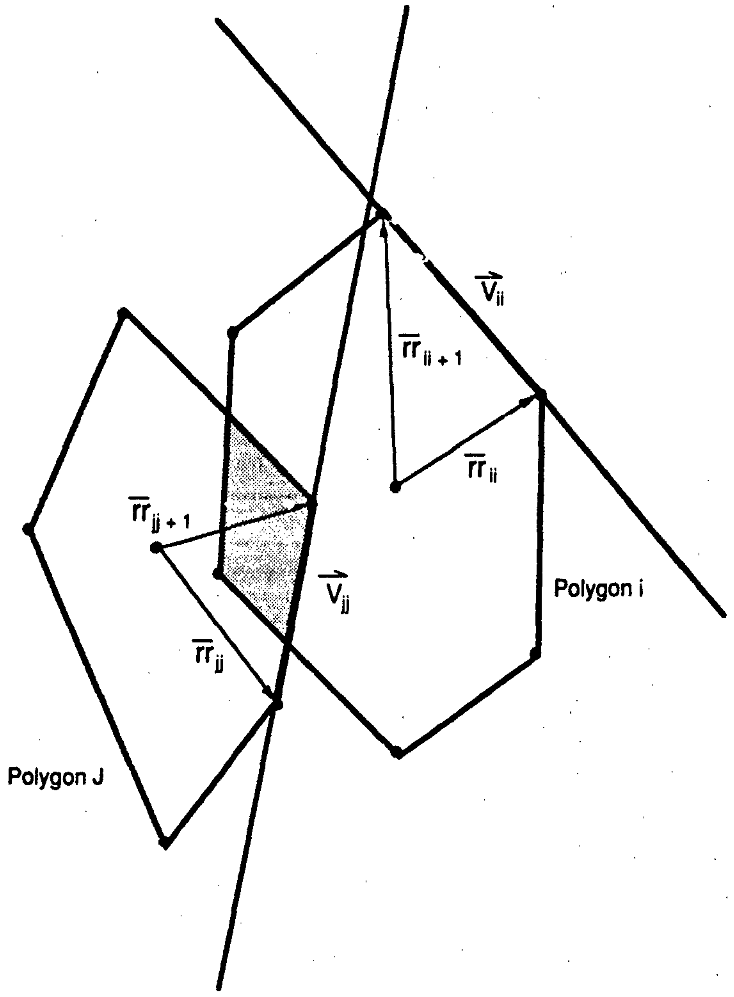


Figure 2. Pair of intersecting convex polygons.

2.4.1 Finding the area of intersections of two convex polygons

A typical contact is shown in Figure 2. One polygon is called the *home* polygon and the other is called the *near* polygon, corresponding to their positions in the list of neighboring pairs compiled in the SEARCH procedure. The array numbers of the *home* and *near* polygons are *i* and *j* respectively.

The procedure used to find the area of intersection is based on an algorithm for finding the intersection of two convex polygons developed by O'Rourke et al. (1982). This algorithm and others are discussed in detail by Preparata and Shamos (1985). The general idea is to advance in a counter-clockwise direction around the perimeters of the two polygons, moving along one side of one polygon at a time. If the advances are made correctly, the intersection points, where the sides of the polygons cross, will be discovered in counter-clockwise order. There are three rules for advancing, which are discussed below.

Let the vertices $ii+1$ and $jj+1$ in Figure 2 be called the current vertices of polygons i and j respectively. Similarly, let the side of polygon i from vertex ii to vertex $ii+1$ and the side of polygon j from vertex jj to vertex $jj+1$ be called the current sides. The current sides of the polygons are defined, with a counter-clockwise sense, by the vectors $\mathbf{V}ii$ and $\mathbf{V}jj$

$$\mathbf{V}ii = \mathbf{rr}(i,ii+1) + \mathbf{rr}(i,ii) \tag{7a}$$

$$\mathbf{V}jj = \mathbf{rr}(j,jj+1) + \mathbf{rr}(j,jj) \tag{7b}$$

where $\mathbf{rr}(i,ii)$ is the vector from the center of mass to vertex ii of polygon i.

A line through the two vertices ii and $ii+1$ of polygon i, extending to infinity in both directions, divides the plane in half. Polygon i lies in the left half-plane. Similarly, a line through the two vertices jj and $jj+1$ of polygon j, extending to infinity in both directions, also divides the plane in half. Polygon j lies on the left side of this line. The intersection of polygons i and j lies in the intersection of the half-planes to the left of each line.

The rules for advance around the perimeters of the polygons depend on whether the current vertices $ii+1$ and $jj+1$ lie inside or outside of the interior (left) half-plane defined by the vector on the current side of the other polygon. The three rules are:

1. If one lies inside and one lies outside, then advance on the polygon whose current vertex lies outside.
2. If both lie inside, then extend the vectors $\mathbf{V}ii$ and $\mathbf{V}jj$ to infinity. Advance on the polygon whose vector tip now lies outside.
3. If both lie outside, then extend the vectors $\mathbf{V}ii$ and $\mathbf{V}jj$ to infinity. Advance on the polygon whose vector tip now lies inside.

An advance is made by incrementing ii or jj. Before each advance, the current sides are tested for intersection.

The three rules for advance are implemented using cross-products. Before defining the cross-products, it is efficient to define an additional vector from vertex ii on polygon i to vertex jj on polygon j

$$\mathbf{V}ij = \mathbf{r}(j) + \mathbf{rr}(j,jj) - \mathbf{r}(i) - \mathbf{rr}(i,ii) \tag{8}$$

where the vector $\mathbf{r}(i)$ defines the location of the center of mass of polygon i. Using the vectors $\mathbf{V}ii$, $\mathbf{V}ij$, $\mathbf{V}jj$ and $\beta = 1$, we define two cross-products

$$xprodij = \mathbf{V}ii \times (\mathbf{V}ij + \beta\mathbf{V}jj) \tag{9a}$$

$$xprodji = \mathbf{V}jj \times (-\mathbf{V}ij + \beta\mathbf{V}ii)\ . \tag{9b}$$

If *xprodij* is positive, then vertex $jj+1$ lies in the interior half-plane defined by $\mathbf{V}ii$. Similarly, if *xprodji* is positive, then vertex $ii+1$ lies in the interior half-plane defined by $\mathbf{V}jj$. If both are positive or both are negative, then, following rule 2 or rule 3, two more cross-products *xprodij'* and *xprodji'* are calculated using eq 9a and b with a large value of β. As with *xprodij*, if *xprodij'* is positive, then the extended vertex $jj+1$ lies in the interior half-plane defined by $\mathbf{V}ii$. Similarly, if *xprodji'* is positive, then the extended vertex $ii+1$ lies in the interior half-plane defined by $\mathbf{V}jj$.

All intersection points will be discovered in, at most, $2[Nv(i) + Nv(j)]$ advances. However, if the values of ii and jj at which the first intersection is discovered are used to initialize ii and jj at the next time step, this may be reduced to approximately $Nv(i)$

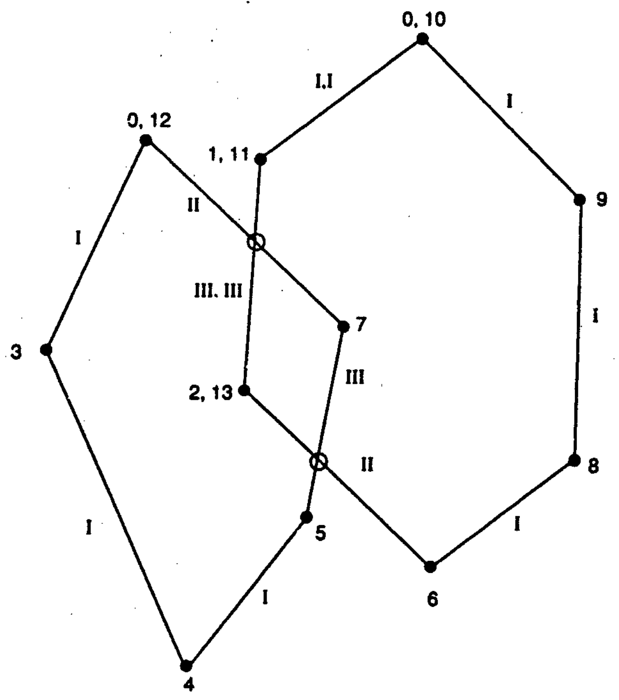


Figure 3. Sequence of advances around the perimeters of the two polygons and the corresponding rules.

+ *Nv(j)* advances. If *Nv(i)* + *Nv(j)* advances are made without finding at least one intersection, then the two polygons must intersect completely or not at all.

Figure 3 shows the two polygons from Figure 2. The numbers adjacent to the vertices correspond to the order of advance. The initial current vertices are those with zeroes adjacent. The Roman numerals I, II and III, adjacent to the sides of the polygons, denote which of the three rules applies to that advance. This algorithm is implemented in the subroutine INTERSECT. The integer counters *nint* and *nadv* keep track of the number of intersections and the number of advances.

2.4.2 Subroutine INTERSECT(i,j)

```
     nint = 0
     nadv = 0
     done = .false.

     dx = x(j)-x(i)
     dy = y(j)-y(i)

     ii = 1
     jj = 1

     Do While (.not. done)

        nadv = nadv+1

C Make vectors and cross-products:
```

```
           iip = 1+mod(ii,Nv(i))
           jjp = 1+mod(jj,Nv(j))

           viix = rx(i,iip)-rx(i,ii)
           viiy = ry(i,iip)-ry(i,ii)

           vjjx = rx(j,jjp)-rx(j,jj)
           vjjy = ry(j,jjp)-ry(j,jj)

           vijx = dx+rx(j,jj)-rx(i,ii)
           vijy = dy+ry(j,jj)-ry(i,ii)

           xprodij = viix*(vijy+vjjy)-viiy*(vijx+vjjx)
           xprodji = vjjx*(-vijy+viiy)-vjjy*(-vijx+viix)
           xprodijo = viix*vijy-viiy*vijx

           xprodjio = -vjjx*vijy+vjjy*vijx

C Check for intersection:

           If (xprodij*xprodijo .lt. 0.0 .and.
        &                        xprodji*xprodjio .lt. 0.0) then

             nint = nint+1

             {store intersection point}

             If (intersection .eq. first intersection) done = .true.

           End If

C Advance on perimeter of polygon i or j depending on cross-products:

           If (xprodij*xprodji .lt. 0.0) then

C One (i or j) is in and one is out:

             If (xprodij .gt. 0.0) then

C j is in, advance on i:

               ii = iip

             Else

C i is in, advance on j:

               jj = jjp

             End If

           Else

C Extend vector Vjj and make a new cross-product ij:
```

```
                mag = 1.0e30

                xprodijp = viix*(vijy+mag*vjjy)-viiy*(vijx+mag*vjjx)

                If (xprodij .gt. 0.0) then

C Both i and j are in:

                    If (xprodijp .gt. 0.0) then

C Extended i is out, advance on i:

                        ii = iip

                    Else

C Extended j is out, advance on j:

                        jj = jjp

                    End If

                Else

C Both i and j are out:

                    If (xprodijp .gt. 0.0) then

C Extended j is in, advance on j:

                        jj = jjp

                    Else

C Extended i is in, advance on i:

                        ii = iip

                    End If
                End If
            End If

            If (nadv .eq. 2*(Nv(i)+Nv(j))      done = .true.

        End Do

        {calculate intersection area if there is any}

        End
```

To calculate the area of the intersection of two polygons, the interior vertices must be known as well as the intersection points. Interior vertices are always signaled by a move following rule 2. The algorithm assures that the vertices of the area of intersection, which consist of one or more pairs of intersection points, and the interior vertices will be discovered in counter-clockwise order. Therefore, it is simple to calculate the area of intersection by dividing it into triangles formed by vertices 1, k and $k + 1$. Similarly, the location of the centroid of the area of intersection is the area weighted sum of the location of the centroids of the triangles.

The line of contact between the polygons is the line connecting the pair of intersection points. In exceptional cases where there are two or more pairs of intersection points, the definition of a line of contact may be somewhat arbitrary. A local coordinate frame is defined with directions normal (n) and tangential (t) to the line of contact. It is oriented such that the normal axis points toward the center of mass of the *home* block.

2.4.3 Calculating the inter-particle force

The force between a pair of intersecting polygons is calculated from the area of intersection, defined in the local **n,t** coordinate frame, and acts at the centroid of the area of intersection. An Elastic–Viscous–Plastic (EVP) normal force model is used with an incremental Mohr–Coulomb tangential force model. The elastic component of the EVP model is proportional to the area of intersection and the viscous component is proportional to the rate of change of the area of intersection. If the sum of the elastic and the viscous components, which act in parallel, is greater than the compressive strength of the material at the contact surface, then the material is assumed to fail plastically. Tensile forces are not considered in this model. Because the plastic deformation is non-recoverable, it is necessary to account separately for total elastic and total combined deformation for each pair of polygons in contact. Total elastic deformation is stored in the variable $Area_e$ and the total combined deformation in the variable *Area*. The tangential force increases incrementally because of slip between the polygons at the contact surface and is limited by the coefficient of friction and the magnitude of the normal force.

The EVP contact force model corrects a problem present in previous polygonal particle simulations (Cundall 1980, Walton 1980), in which the viscous damping force is proportional to the normal component of the relative velocity between the polygons at the point of contact. This damping method is unappealing in vertex/vertex and vertex/side contacts because the damping force is a maximum when the area of intersection is a minimum, namely, when the vertex of one polygon first contacts the other polygon. In contrast, a viscous damping force proportional to the rate of change of the area of intersection is small when the point first contacts the other polygon. However, the latter damping method does not dissipate enough energy in point/point and point/side contacts to model highly inelastic behavior. The essential ingredient in modeling inelastic behavior is the plastic limit on the stress across the plane of contact, which qualitatively accounts for the initial crushing that takes place in vertex/vertex and vertex/side contacts. In the present model, plastic yielding does not alter the shape of the polygons involved. Following rebound, the polygons regain their original undeformed shapes. However, except for increasing complexity, nothing prevents the polygon's shape from being recalculated as it deforms.

The contact force model is based on pseudo-material properties of the polygons—a stiffness k_{ne}, a viscous damping constant k_{nv}, a compressive strength σ_c and a coefficient of friction μ. The dimensions of k_{ne} and k_{nv} are force per unit area and force-time per unit area respectively. Ideally, the stiffness should be based on the modulus of elasticity of the material being modeled. For stability, the time step used is a small fraction (typically 1/10) of the period of the highest frequency expected in the simulation. This is approximately the frequency of free oscillation $(1/2\pi)(k_{ne}/m)^{1/2}$, where m is the mass of the smallest polygon divided by the nominal maximum contact width.

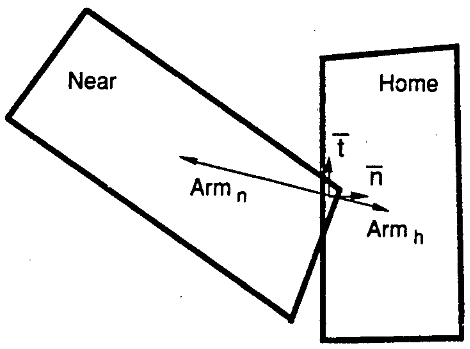


Figure 4. Interparticle contact force model.

In problems where the elastic properties of the material must be accurately modeled, for example in the propagation of elastic waves through an assembly of particles, a realistic stiffness must be used. However, in problems in which the forces are a function of gravity and the geometric angularity of the blocks, such as in the ice ridging and shearbox simulations shown in sections 4.1–4.3, the stiffness may be lessened to increase the time step and thus make it possible to simulate larger or longer experiments. The amount depends on the magnitude of the expected forces and the amount of overlap that can be tolerated between blocks in contact without compromising their geometric integrity.

The viscous damping constant k_{nv} is chosen as a fraction of the critical damping value $2(k_{ne}\,m)^{1/2}$ based on the stiffness and the minimum mass per meter of contact width, m, defined above. The compressive strength σ_c of the material must be scaled if the stiffness is reduced in the interest of computational speed.

A pair of polygons in contact is shown in Figure 4. The *home* and *near* polygons are denoted by the subscripts h and n respectively. The moment arms from the centroid of the area of intersection to the centers of mass of the polygons are

$$\mathbf{arm}_h = \mathbf{r}_c - \mathbf{r}(nh) \tag{10a}$$

$$\mathbf{arm}_n = \mathbf{r}_c - \mathbf{r}(nn) \tag{10b}$$

where nh and nn are the numbers of the *home* and *near* polygons and the vector $\mathbf{r}_c$ defines the location of the centroid of the area of intersection.

The relative velocity of the *home* polygon with respect to the *near* polygon at the point of contact is

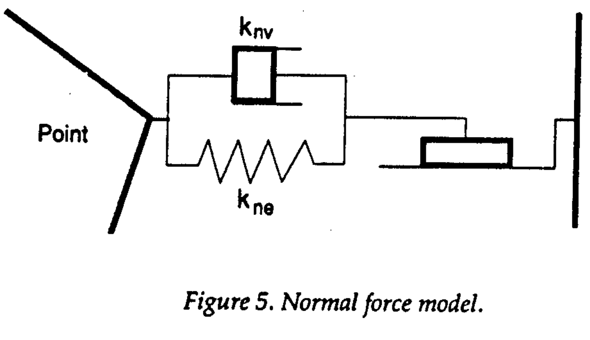


Figure 5. Normal force model.

$$\mathbf{V}_{h/n} = \mathbf{V}_h + \omega_h \times \mathbf{arm}_h - \mathbf{V}_n - \omega_n \times \mathbf{arm}_n \,. \tag{11}$$

A simplified representation of the normal component of the contact force model is shown in Figure 5. The details of the model are presented in the following equations. It is worth noting that, although the spring and dashpot in the figure are linear, the area of intersection, and thus, the elastic and viscous force components, change nonlinearly, especially in vertex/side and vertex/vertex contacts. The change in the total area of intersection during the previous time step Δt is

$$\Delta Area^n = Area^n - Area^{n-1} \tag{12}$$

where the superscript n denotes the current time. It is initially assumed that the incremental deformation $\Delta Area^n$ was elastic. In this case the elastic component of the normal force exerted on the *home* particle is

$$F_{ne}{}^n = k_{ne}\,(Area_e{}^{n-1} + \Delta Area^n) \tag{13}$$

where k_{ne} is the normal elastic stiffness. The average rate of change of the area of intersection during the previous time step is

$$(\Delta Area/\Delta t)^{n-1/2} = \Delta Area^n/\Delta t \,. \tag{14}$$

The current rate of change of the area of intersection is approximately

$$(\Delta Area/\Delta t)^n = {}^3/_2\,(\Delta Area/\Delta t)^{n-1/2} - {}^1/_2\,(\Delta Area/\Delta t)^{n-3/2} \,. \tag{15}$$

The viscous component of the normal force exerted on the *home* polygon is then

$$F_{nv}{}^n = k_{nv}\,(\Delta Area/\Delta t)^n \tag{16}$$

where k_{nv} is the normal viscous damping constant. The maximum compressive force that the contact may support is the product of the compressive strength of the material σ_c and the breadth of the contact surface $length_c{}^n$. This maximum will be referred to as the plastic limit force F_p

$$F_p{}^n = \sigma_c\, length_c{}^n \,. \tag{17}$$

The breadth of the contact surface is the distance between the intersection points in Figure 3 above. If the sum of the elastic and viscous components of the normal force is less than the plastic limit force, then the normal force is equal to that sum

$$F_n{}^n = F_{ne}{}^n + F_{nv}{}^n \tag{18}$$

and the incremental change in the total area (eq 12) is added to the elastic area

$$Area_e{}^n = Area_e{}^{n-1} + \Delta Area^n \,. \tag{19}$$

If the sum of the elastic and viscous components of the normal force is greater than or equal to the plastic limit force, then the normal force is equal to the plastic limit force

$$F_n{}^n = F_p{}^n \,. \tag{20}$$

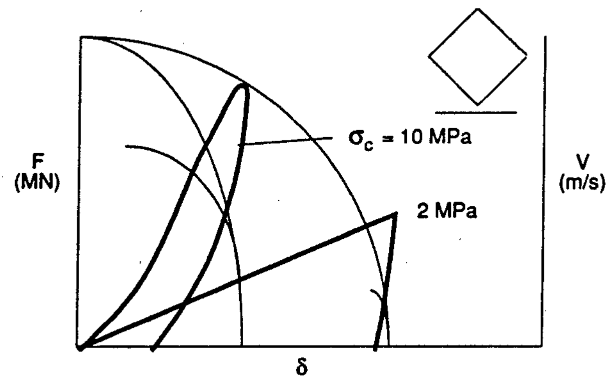
In this case, since there is motion across the spring and dashpot, part of the incremental increase in the total deformation $\Delta Area^n$ will be plastic and the remainder will be elastic. The incremental increase in the elastic deformation $\Delta Area_e^n$ is calculated by solving a differential equation in finite difference form derived from the balance of the plastic limit force (eq 17) and the sum of the elastic (eq 13) and viscous forces (eq 16)

$$(\Delta Area_e/\Delta t)^n = (1/k_{nv})(F_p^n - k_{ne}\, Area_e^n)\,. \tag{21}$$

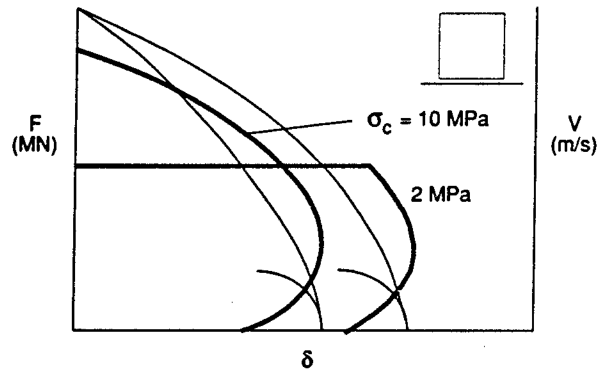
In the simulation, eq 21 is solved for $Area_e^n$ using a Runge–Kutta or other approximate solution. Finally, since tensile forces are not considered, the normal force on the *home* polygon must be greater than or equal to zero (the n direction points toward the *home* particle).

The mechanics of the EVP normal contact force model are qualitatively illustrated in Figure 6, which shows the results of numerical experiments in which a square block was released from a position just above a flat surface with an initial downward velocity. Figure 6a pertains to a vertex/side contact in which the block falls vertex first. Figure 6b pertains to a side/side contact in which the block falls edge first. The stiffness and viscosity were held constant. Each figure shows results for compressive strengths $\sigma_c = 2$ and 10 MPa. The velocity trace begins at the maximum of the left axis in each case. The penetration distances are plotted to different scales in Figures 6a and 6b.

In each figure, the contact force traces for the two values of compressive strength are qualitatively different. At the higher value of compressive strength, the contact



a. For a point on side contact.



b. For a side on side contact.

Figure 6. Normal contact force versus depth of penetration δ.

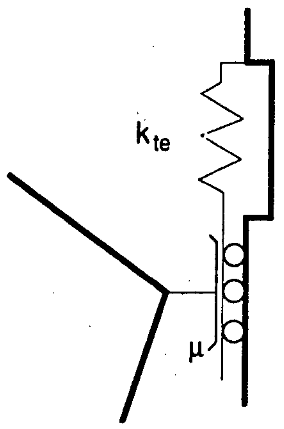


Figure 7. Tangential force model.

forces are elastic–viscous (eq 18). At the lower value of compressive strength, the contact forces during loading are plastic (eq 17).

A representation of the tangential contact force model is shown in Figure 7. The tangential force F_t increases because of incremental slip between the polygons at the contact surface in the tangential direction (Walton 1980). The incremental slip occurring between the polygons from time step to time step creates the tangential force by compressing the spring in the figure. The magnitude of the tangential force is not allowed to exceed μF_n, where μ is the coefficient of friction. The tangential force on the *home* polygon is

$$F_t^{n} = F_t^{n-1} - k_{te}\,\Delta t(\mathbf{V}_{h/n} \bullet \mathbf{t})^{n-1/2} \tag{22}$$

where k_{te} is the tangential elastic stiffness. The forces on the *home* polygon in the global **x,y** coordinate frame are

$$F_x^{n} = n_x F_n^{n} - n_y F_t^{n} \tag{23a}$$

$$F_y^{n} = n_y F_n^{n} + n_x F_t^{n} \tag{23b}$$

where n_x and n_y are the components of the normal vector in the local coordinate frame. The force on the *near* polygon is equal and opposite. The moments M_h^n on the *home* particle and M_n^n on the *near* polygon are

$$M_h^{n} = \mathbf{arm}_h^{n} \times \mathbf{F}^{n} \tag{24a}$$

$$M_n^{n} = \mathbf{arm}_n^{n} \times -\mathbf{F}^{n}\,. \tag{24b}$$

2.5 Equations of particle motion

Once the forces exerted on each particle by surrounding particles have been calculated, the equations of motion for each particle may be solved and time advanced one step. The equations of motion are derived from a Taylor series expansion about the current time (Walton 1980). The x component of velocity $u^{n+1/2}$ of particle i is

$$u_i^{n+1/2} = u_i^{n-1/2} + \Delta t\, F_{xi}^{n}/mass_i\,. \tag{25}$$

The position of particle i at time n+1 is then

$$x_i^{n+1} = x_i^n + \Delta t\, u_i^{n+1/2}\,. \tag{26}$$

These expressions are second-order accurate with velocity-dependent forces. Similar equations are used for the y component of velocity v, the y position, the angular velocity ω and the orientation θ. Again, for stability, the time step Δt is a small fraction (typically 1/10) of the reciprocal of the frequency of free oscillation $(1/2\pi)(k_{nc}/m)^{1/2}$, where m is the mass of the smallest particle.

It is efficient to calculate and store the positions of the particle vertices. The x and y components of the vector from the center of mass of particle i to vertex j are

$$r_x^n(i,j) = r(i,j)\cos[\theta^n(i) + \alpha(i,j)] \tag{27a}$$

$$r_y^n(i,j) = r(i,j)\sin[\theta^n(i) + \alpha(i,j)]\,. \tag{27b}$$

2.6 Calculating the energetics of a system

The work performed on a system during the course of a numerical experiment is balanced by changes in the potential energy, the kinetic energy and the energy stored in elastic deformation, and by energy dissipation. The dissipative mechanisms consist of frictional contacts between sliding blocks and inelasticity and crushing at the point of contact, modeled by the viscous–plastic parts of the normal contact force model.

The rate of work performed on a system is the sum of the dot products of the forces on, and the velocity of, each boundary particle in the system. In general, either the force or the velocity of the boundary particles may be specified.

The rate of change in the potential energy of a system is the sum of the dot products of the body forces on, and the velocity of, each particle in the system. An expression for the change in the potential energy of a system in which the body forces are gravity and buoyancy is

$$\Delta PE = g\,\Delta t \sum_n \sum_i \left(w_i^n \rho_i A_i - w_{is}^n \rho_w A_{is}^n\right). \tag{28}$$

The two summations in eq 28 are over each particle (i) at each time step (n). A_i is the area and w_i is the vertical component of the velocity of particle i. A_{is} is the submerged area and w_{is} is the vertical component of the velocity of the centroid of the submerged area of particle i. ρ_i is the particle density and ρ_w is the water density.

The frictional dissipation Φ_f is the sum of the work performed by the tangential force at each point of contact among the particles in the system

$$\Phi_f = \Delta t \sum_n \sum_i (\mathbf{F}\cdot\mathbf{t})_i^n (\mathbf{V}_{h/n}\cdot\mathbf{t})_i^n. \tag{29}$$

The two summations in eq 29 are over all of the pairs of particles in contact (i) at each time step (n). The relative velocity is given by eq 11 and the tangential force component is given by eq 22. Similarly, the inelastic dissipation Φ_i, attributable to viscous damping and plastic deformation, is the sum of the net work performed by the normal force at each point of contact among the particles in the system

$$\Phi_i = \Delta t \sum_n \sum_i (\mathbf{F}\cdot\mathbf{n})_i^n (\mathbf{V}_{h/n}\cdot\mathbf{n})_i^n. \tag{30}$$

Again the two summations in eq 30 are over all of the pairs of particles in contact at each time step. The relative velocity is given by eq 11 and the normal force component is given by eq 18 or 20. The inelastic dissipation includes energy remaining in elastic deformation at the end of an experiment. It may be calculated if desired using the equation $E = \sum_i k_{ne}\, Area_{ei}$, where $Area_{ei}$ is the elastic area remaining at contact i.

The change in the kinetic energy of the system is found by calculating the translational and rotational kinetic energy of each particle at the beginning and end of the experiment.

2.7 Simple breakage routine for slender blocks

Because of the tendency for polygonal blocks to form stable load-bearing structures by interlocking or stacking, some form of breakage mechanism may be necessary for realism and for the relief of forces that would otherwise continue to grow unchecked.

A simple breakage model suitable for slender, rigid blocks was used in simulated ridge growth experiments (Hopkins et al. 1991) and in simulated shear box experiments (Hopkins and Hibler 1991a). In these experiments, the ice blocks were quadrilateral with their long sides parallel as shown in Figure 8. Flexure in the x,y plane caused the blocks to break. A typical block is shown in Figure 8 with the contact forces imposed by neighboring blocks.

At regular intervals during the simulation, the d'Alembert force and torque caused by the contact forces and body forces on each rigid block are calculated. The addition of the moment of the calculated force and the calculated torque to the moment of the other forces on the block reduces the calculation of the moment at a point in the block to a statics problem. The y component of acceleration, attributable to the contact forces F_{y_i} and the body forces F_b on the block, is

$$a_y = \left(F_b + \sum_i F_{y_i}\right) / mass. \tag{31}$$

The rotational acceleration about the z-axis is

$$\alpha = \left(M_b + \sum_i F_{y_i} x_i\right) / J_p \tag{32}$$

where J_p is the polar moment of inertia about the z-axis. A method for calculating the polar moment of inertia of an arbitrary polygon is outlined in section 2.8. M_b is the moment of the body forces on the particle. In a uniform gravitational field, this term would be zero, but if, for example, the block were partially submerged in water, it would be necessary to integrate over the length of the block to find M_b.

The moment at point X in the block is

$$M_X = -\sum_i min[(x_i - X),0]\, F_{y_i} + \rho \int_{xL}^{X} (x - X)(-\alpha x - a_y)\, dx\,. \tag{33}$$

The moment M_X is calculated at small intervals along the x-axis of the block. If the moment at some point exceeds the flexural strength of the block, then the block is broken at that point. The piece containing the center of mass of the original block retains the same index. A new index $N + 1$ is assigned to the other piece. The new

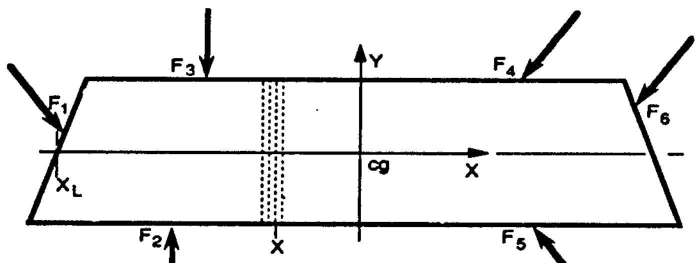


Figure 8. Calculation of the moments in a block of ice rubble.

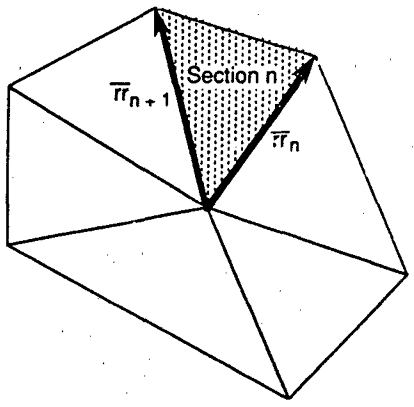


Figure 9. Calculation of the mass and center of mass of a convex polygon.

center of mass of each of the pieces is calculated along with the new radii and angles to the vertices defining the shapes. The new mass and polar moment of inertia of each block are calculated and both blocks are given the translational and rotational velocities and angular position of the original block. Algorithms for determining the mass, the center of mass and the polar moment of inertia of polygonal blocks will be discussed in the next section of this work. In passing, it is noted that this approach could be used to model shear failure as well.

2.8 Calculating the mass, center of mass and polar moment of inertia of a convex polygon

The mass and the location of the center of mass of a convex polygon may be found by using the vectors from some arbitrary point in the interior of the polygon to each vertex to divide the polygon into triangular sections as shown in Figure 9. The area and centroid of each triangular section are then calculated individually. The area of the triangle defined by the vectors **rr**(n) and **rr**(n + 1) is

$$Area_n = {}^1/_2 \mid \mathbf{rr}(n) \times \mathbf{rr}(n+1) \mid . \tag{34}$$

The centroid of the triangular area is located at

$$\mathbf{r}_{c\,n} = {}^1/_3\,[\mathbf{rr}(n) + \mathbf{rr}(n+1)] \tag{35}$$

with respect to the arbitrary interior point.

The same convex polygon may be divided again into triangular sections by drawing lines from its center of mass to each vertex. The center is a vertex of each triangular segment. The moment of inertia of the polygon about an axis perpendicular to the polygon through its center is the sum of the moments of inertia of each triangle about the same axis. Figure 10 shows a single triangular segment of the polygon. We seek an expression for the moment of inertia of the triangle about an axis perpendicular to the triangle at vertex O.

A line through O perpendicular to the opposite side divides the triangle into two right triangles Δ_1 and Δ_2. The polar moment of inertia of a right triangle about its center of mass is

$$J = (bh/36)\,(b^2 + h^2) \tag{36}$$

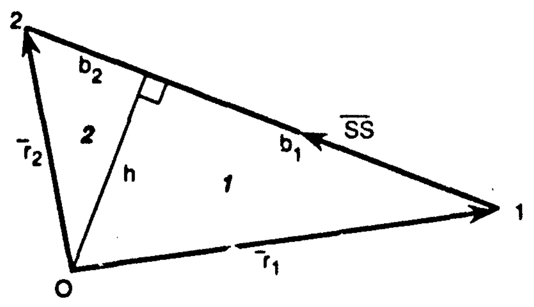


Figure 10. Triangular segment of a convex polygon.

where b and h are the lengths of the perpendicular sides of the right triangle. The moment about the point O by the parallel axis theorem is

$$J_o = J + Ad^2 \tag{37}$$

where $A = 1/2bh$ is the area of the triangle and d is the distance from the center of mass to the point O.

In Figure 10 the unit vector **ss**, defining the direction of the side opposite O, is given by

$$\mathbf{ss} = (\mathbf{rr}_2 - \mathbf{rr}_1)/\mid \mathbf{rr}_2 - \mathbf{rr}_1 \mid . \tag{38}$$

The lengths b_1, b_2 and h are given in terms of **ss** as

$$b_1 = \mid \mathbf{ss} \bullet \mathbf{rr}_1 \mid \tag{39a}$$
$$b_2 = \mid \mathbf{ss} \bullet \mathbf{rr}_2 \mid \tag{39b}$$
$$h = \mid \mathbf{ss} \times \mathbf{rr}_1 \mid = \mid \mathbf{ss} \times \mathbf{rr}_2 \mid . \tag{39c}$$

The moment of Δ_1 about its center of mass, determined using eq 36 and 39a,c, is

$$J_1 = {}^1/_{36}\, b_1 h\, (b_1{}^2 + h^2)\,. \tag{40}$$

The area of Δ_1 is $1/2\ b_1h$ and the distance from the center of mass to point O is

$$d_1 = {}^1/_3\, (b_1{}^2 + 4h^2)^{1/2}\,. \tag{41}$$

Substituting eq 39a,c, 40 and 41 into eq 37, and rearranging terms, yields the polar moment of inertia of triangle 1 about O

$$J_{1o} = {}^1/_{12}\, b_1 h\, (b_1{}^2 + 3h^2)\,. \tag{42}$$

The expression for J_{2o}, the moment of inertia about O of triangle 2 in Figure 10, is similar. The moment of inertia about O of the complete triangle is the sum of J_{1o} and J_{2o}. The polar moment of inertia of an arbitrary convex polygon is found by adding the moments of inertia of each triangular segment about the centroid of the polygon.

2.9 Troubleshooting

Problems with the polygonal simulation fall into two general categories:

1. Dynamic instabilities in the interparticle force modeling.

2. Problems in the search routine from unobserved contacts.

Dynamic instabilities are caused by a time step that is too large for the elastic constant and viscous damping constant being used. The fundamental frequency in the simulation is approximately the frequency of free oscillation of the elastic component of the normal force (eq 13).

In side on side contacts, the area is proportional to the depth of penetration, while in point on side and point on point contacts, it is proportional to the square of the depth. For simplicity, the linear case is assumed. The frequency of free oscillation of an undamped linear oscillator is $(1/2\pi)(k_{ne}/m)^{1/2}$. If the system is not overdamped, a time step Δt, which is a small fraction (typically 1/10) of the reciprocal of the frequency, calculated with m equal to the mass of the smallest polygon in the system divided by the nominal maximum contact width, is a conservative starting point.

Unobserved contacts have several simple causes. Contacts may be missed if the search neighborhood (see Fig. 1) about each particle is not large enough to contain the centers of all possible neighbors. The solution is to increase the size of the neighborhood by increasing the cell size in the grid overlying the flow domain.

Unobserved contacts may also stem from the strategy of not calling the SEARCH subroutine at every time step. If the system is rapidly deforming, two polygons that were separated by a distance greater than ε during the previous search may move into contact. Since the pair of polygons is not in the list of neighboring pairs, the contact will remain undetected until the next search is performed. By that time their area of intersection may be large. When they are finally detected, the force between them will be huge and the aftermath will resemble an explosion. The solution is to increase ε or decrease the interval between SEARCH calls.

The interval between SEARCH calls must be chosen with an eye to the amount of deformation that the system is likely to undergo in the interval and the parameter ε, the maximum distance between the pairs of particles placed in the list of near neighbors. For example, if ε is 0.01 m and the nominal relative velocity between particles is 1 m/s, then the search routine must be called at least every 0.01 s. It is suggested that the search routine be called at intervals of no more than 50 time steps. The parameter ε should be larger than product of the time interval between global searches and the estimated maximum relative velocity between particles, including rotation. It is inefficient to make ε larger than necessary, as this will increase the number of pairs in the near-neighbor list, all of which must be tested for contact at each time step. These guidelines are necessarily arbitrary. Experience is the only real guide.

3. ALGORITHM FOR MODELING GLUED JOINTS BETWEEN DISCRETE POLYGONS

The dynamics of river jamming and sea ice ridging processes depend on the dynamics of rubble accumulation and floating plates of intact ice. The rubble dynamics are well modeled by existing two-dimensional simulations of polygonal blocks, such as the one described in section 2 of this work and by Walton (1980). In this section, a dynamic simulation technique is developed, which is consistent with the particle simulation and is suitable for two-dimensional modeling of a body, such as an intact ice sheet, composed of discrete, rigid, convex polygonal elements.

The algorithm is developed for the general case of a two-dimensional material composed of non-uniform, convex polygonal elements. This general algorithm has been further developed for the case of equilateral triangular elements for the study of fracture of granular materials by Potapov et al. (in prep.). The algorithm was used by Hopkins (1991) to model a beam composed of uniform rectangular elements.

As the modeled material deforms, internal tensile, compressive and shear forces

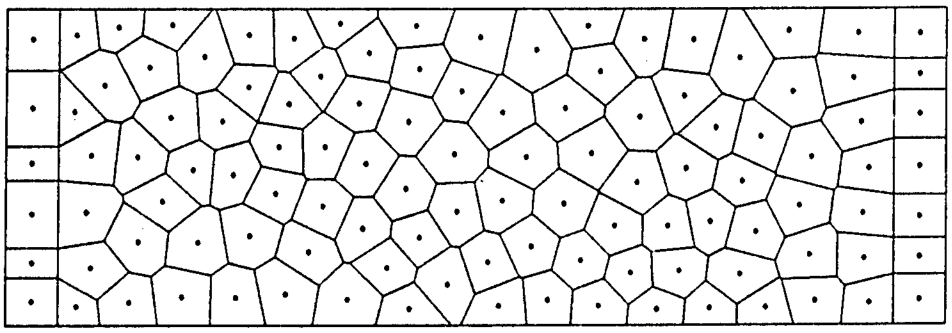


Figure 11. Specimen composed of discrete, convex polygonal blocks.

are created. A viscous–elastic force model is used, which is similar to the one described in section 2.4.3. If the tensile or compressive stresses in a glued joint exceed the strength of the material, a crack is initiated in the joint. The crack propagates along the joint at a finite speed, creating a discontinuity in the material.

3.1 Glued joint simulation

Figure 11 shows a rectangular specimen composed of discrete, rigid polygonal elements or blocks. The joints between adjacent blocks are capable of supporting both tensile and compressive forces in the direction normal to the glued faces and shear forces in the tangential direction. The properties of the glued joints are determined by parameters defining the stiffness and damping characteristics of the material. Conceptually, the glued joint simulation method differs from the discrete block simulation method described in previous sections of this report in the modeling of tensile forces and in the fixed spatial relationship between adjacent blocks.

The edges of adjoining blocks are assumed to be joined by viscous–elastic fibers. As the blocks move relative to one another because of deformation of the specimen, the fibers stretch or compress. A pair of blocks that have rotated relative to one another is shown in Figure 12. The vectors δ_1 and δ_2 shown in the figure represent fibers in tension and compression, respectively. The vectors are defined by the position of a point on block 2 with respect to the adjacent point on block 1 in the undeformed state. It is assumed that relative deformation measured by δ is small compared to the size of the blocks. The forces on each block are assumed to be equal and opposite.

A local η,ζ coordinate frame is shown in Figure 13, with its origin at the center of the face of block 1. The length of the joint is L. The η direction is outwardly

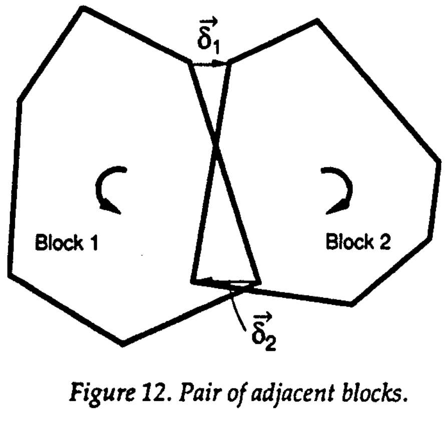


Figure 12. Pair of adjacent blocks.

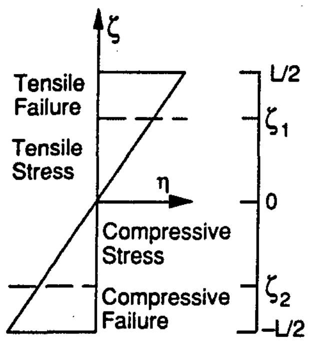


Figure 13. Local coordinate frame.

perpendicular to the face of block 1 and the ζ direction is tangential. The η and ζ components of δ(ζ) will be denoted $\delta_\eta(\zeta)$ and $\delta_\zeta(\zeta)$. Each varies linearly between the values at the ends of the joint.

The elastic normal stress in a fiber located at ζ on the glued face is

$$\sigma(\zeta) = k_{\eta\,e}\,\delta_\eta(\zeta) \tag{43}$$

where $k_{\eta\,e}$ is the stiffness of the material in the η direction. In a beam formed by joining rectangular blocks of width w, $k_{\eta\,e} = E/w$, where E is Young's modulus. The total elastic force in the η direction is found by integrating eq 43 along the length of the joint

$$F_{\eta\,e} = k_{\eta\,e}\int_{-L/2}^{+L/2} \delta_\eta(\zeta)\,d\zeta\,. \tag{44}$$

An integral part of the glued joint simulation is the capacity to model flexural failure. Failure occurs if the stress (eq 43) in the fiber at either end of the joint exceeds the compressive or tensile strength of the material. In this work compressive failure is not considered.

In the glued joint simulation, the crack propagates at a constant specified velocity. No force is exerted across the cracked part of a joint failing in tension. Therefore, if the joint is undergoing failure, the integration limits in eq 44 are reduced so that integration is only carried out across the intact region.

$$F_{\eta\,e} = k_{\eta\,e}\int_{\xi_2}^{\xi_1} \delta_\eta(\zeta)\,d\zeta \tag{45}$$

where ζ_1 and ζ_2 are $+L/2$ and $-L/2$, respectively, in the absence of a crack and one or the other is set equal to the crack tip position in the presence of a crack.

The integrals in eq 45 and in subsequent equations are evaluated by expressing $\delta_\eta(\zeta)$ in terms of its slope and ζ-intercept using the values of $\delta_\eta(\zeta)$ at ζ_1 and ζ_2 in Figure 13. Similarly, $\dot{\delta}_\eta(\xi)$ may be expressed in terms of the time derivative of the slope and ζ-intercept using the previously calculated values of the slope and ζ-intercept. In the cracked regions, the elastic force across a crack tip in tension drops immediately to zero. However, the elastic force across the intact part of the joint continues. The stored elastic force is released smoothly during fracture through eq 45. An elastic compressive force that persists after complete fracture will be modeled without loss of continuity by the equations for contact forces in section 2.4.3.

A dissipative or viscous force is assumed to exist that is proportional to the rate of deformation of the material. The dissipative force on block 1 in the η direction is

$$F_{\eta\,v} = k_{\eta\,v} \int_{\xi_2}^{\xi_1} \dot{\delta}_\eta(\zeta)\, d\zeta\,. \tag{46}$$

The dissipative force exists only across the unfractured part of the joint.

The elastic force $F_{\eta\,e}$ and the damping force $F_{\eta\,v}$ exert moments about the midpoint of the glued joint because of the linear variation of the forces along the joint. The moment of the elastic force on block 1 about the midpoint is

$$M_{\eta\,e} = -k_{v\,e} \int_{\xi_2}^{\xi_1} \zeta\delta_\zeta(\zeta)\, d\zeta \tag{47}$$

where the integral is evaluated as in eq 45. The moment of the damping force on block 1 about the midpoint is

$$M_{\eta\,v} = -k_{\eta\,v} \int_{\xi_2}^{\xi_1} \zeta\dot{\delta}_\zeta(\zeta)\, d\zeta\,. \tag{48}$$

Shear forces are created on the glued joint by the displacement of block 2 with respect to block 1 in the tangential (ζ) direction. In theory, the distribution of shear forces on a joint is not uniform. In the present case, the blocks are assumed to be rigid and, therefore, the shear force, which is constant over the face, should be interpreted as the average of the true non-uniform shear force. The elastic component of the shear force in the ζ direction in the unbroken part of the joint is

$$F_{\zeta\,e} = k_{\zeta\,e} \int_{\xi_2}^{\xi_1} \delta_\zeta(\zeta)\, d\zeta. \tag{49}$$

The integral in eq 49 is evaluated by expressing $\delta_\zeta(\zeta)$ in terms of its slope and ζ-intercept using the values of $\delta_\zeta(\zeta)$ at ζ_1 and ζ_2 in Figure 13.

The dissipative component of the shear force in the ζ direction is

$$F_{\zeta\,v} = k_{\zeta\,v} \int_{\xi_2}^{\xi_1} \dot{\delta}_\zeta(\zeta)\, d\zeta\,. \tag{50}$$

The integral in eq 50 is evaluated by expressing $\delta_\zeta(\zeta)$ in terms of the time derivative of the slope and ζ-intercept using the previously calculated values of the slope and ζ-intercept.

The component forces on block 1 in the local η,ζ coordinate frame are added before being converted to the global x,y coordinate frame

$$F_{x\,1} = \eta_x\,(F_{\eta\,e} + F_{\eta\,v}) - \eta_y\,(F_{\zeta\,e} + F_{\zeta\,v}) \tag{51a}$$

$$F_{y\,1} = \eta_y\,(F_{\eta\,e} + F_{\eta\,v}) + \eta_x\,(F_{\zeta\,e} + F_{\zeta\,v}). \tag{51b}$$

The forces on block 2 are equal and opposite.

The elastic and viscous forces in the η and ζ directions exert moments about the block centers of mass, which depend on the vector from the block centers to the midpoint of the face of the joint. These moments are added to the moments $M_{\eta\,e}$ (eq 47) and $M_{\eta\,v}$ (eq 48) about the midpoint of the joint. The total moments on block 1 and block 2 from all of the forces are

$$M_1 = (\mathbf{r}_o)_1 \times \mathbf{F}_1 + M_{\eta\,e} + M_{\eta\,v} \tag{52a}$$

$$M_2 = -[(\mathbf{r}_o)_2 \times \mathbf{F}_1 + M_{\eta\,e} + M_{\eta\,v}] \tag{52b}$$

where $(\mathbf{r}_o)_1$ and $(\mathbf{r}_o)_2$ are the vectors from the centers of mass of blocks 1 and 2 to the midpoint of the face of the joint. The internal forces and moments exerted on each block by adjoining glued blocks are added to the forces exerted on the blocks by contacts with surrounding unconnected blocks.

3.2 Glued joint energetics

Elastic energy stored in the glued joint and energy dissipated by viscous damping are components in the energetics of the overall simulation. The energy picture during fracture is somewhat complicated. The elastic energy stored in tension in a glued joint undergoing fracture is lost. The elastic energy stored in compression may either be unloaded during failure or unloaded after complete failure by the contact force equations in section 2.4.3.

The rate of elastic energy storage, in the normal direction, perpendicular to the glued faces, is found by multiplying the integrand in eq 45 by $\dot{\delta}_\eta(\zeta)$, the η component of the rate of deformation of the fibers on the glued face

$$\Phi_{\eta\,e} = k_{\eta\,e} \int_{\xi_2}^{\xi_1} \delta_\eta(\zeta)\, \dot{\delta}_\eta(\zeta)\, d\zeta\,. \tag{53}$$

The rate of elastic energy storage in the tangential direction in the unfractured region is found by multiplying the integrand in eq 50 by $\dot{\delta}_\zeta(\zeta)$, the ζ component of the rate of deformation of the fibers on the glued face

$$\Phi_{\zeta\,e} = k_{\zeta\,e} \int_{\xi_2}^{\xi_1} \delta_\zeta(\zeta)\, \dot{\delta}_\zeta(\zeta)\, d\zeta\,. \tag{54}$$

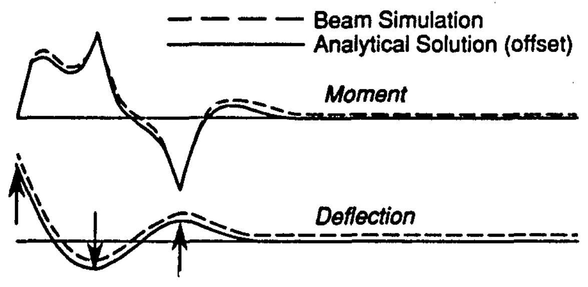


Figure 14. Simulation of a floating beam compared to analytical solutions.

Similarly, the energy dissipation rates attributable to viscous damping in the normal and the tangential directions in the unfractured region are

$$\Phi_{\eta\,v} = k_{\eta\,v} \int_{\xi_2}^{\xi_1} \dot{\delta}_{\eta}^{2}(\zeta)\, d\zeta \tag{55}$$

and

$$\Phi_{\zeta\,v} = k_{\zeta\,v} \int_{\xi_2}^{\xi_1} \dot{\delta}_{\zeta}^{2}(\zeta)\, d\zeta\,. \tag{56}$$

3.3 Discussion

Previous efforts devoted to explicit modeling of sea ice (Parmerter and Coon 1972, Hopkins et al. 1991) have made use of analytical solutions for an equilibrated beam on an elastic foundation (Hetenyi 1946). Hopkins (1991) applied the glued joint approach to modeling a floating ice sheet. Figure 14, which is from that work, shows a comparison between a floating beam simulated using the glued joint approach described above and the analytical solution for a floating beam subjected to three point loads of 1 kN, indicated by the arrows. The figure shows the static deflection and moment with the analytical solutions offset for visibility.

A dynamic simulation of a cantilever beam was compared with the linear elastic theory of Timoshenko (1955). The frequency of oscillation of the simulated beam was within 2% of the theoretical prediction.

Ultimately, the glued joint beam simulation will be used to model an intact ice cover in river jamming and sea ice ridging experiments. Snapshots of a sea ice ridging simulation are shown in the Figure 20. The details of that simulation are described by Hopkins and Hibler (1991b).

In principal, extension of the glued joint method to simulate two- and three-dimensional arrays of elements should be straightforward. In two or three dimensions, pressure on one pair of opposing faces of a block would cause a response on the other pairs of opposing faces, which would be a function of Poisson's ratio. A three-dimensional simulation using cubic blocks would allow modeling of the deformation in an ice sheet in conjunction with a three-dimensional particle simulation using polyhedral blocks.

4. MODELING SEA ICE WITH THE POLYGONAL SIMULATION

4.1 Model shear box experiments

A series of numerical experiments with a simulated two-dimensional shearbox is described by Hopkins and Hibler (1991a). The object of these experiments was to examine the effects of load, friction and breakage on the shear strength of ice rubble. The samples were prepared by loosely filling the shear box shown in Figure 15 with blocks having a length-to-thickness distribution characteristic of Baltic pressure ridges. The lid was placed over the rubble, the load was applied, and the system was allowed to come to rest.

In the experiments, the lower part of the shear box moved at a constant horizontal speed with respect to the fixed upper part. The lid was free to move vertically to balance the applied load *N* and the forces exerted by the ice rubble on its lower surface. The lid was not allowed to rotate or to move laterally. The experiments were run without gravity to remove the uncertainty as to the true load on the failure surface caused by the overburden. The horizontal force *S* required to move the lower part of

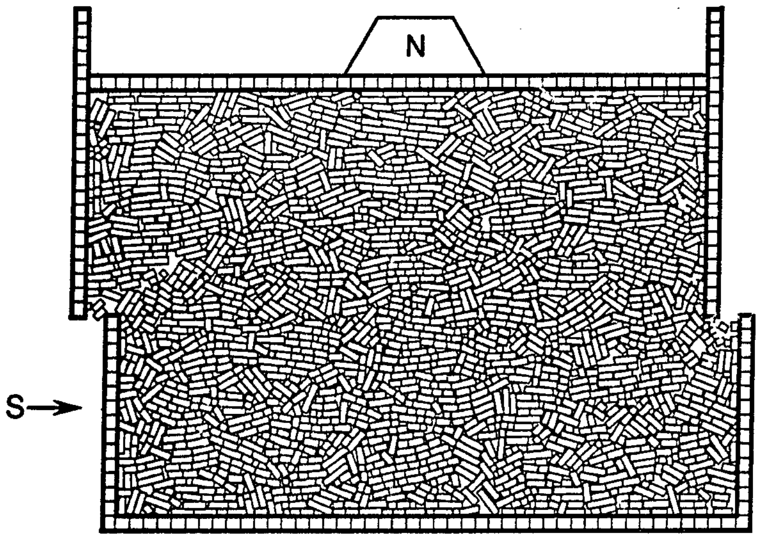


Figure 15. Simulated direct shear box.

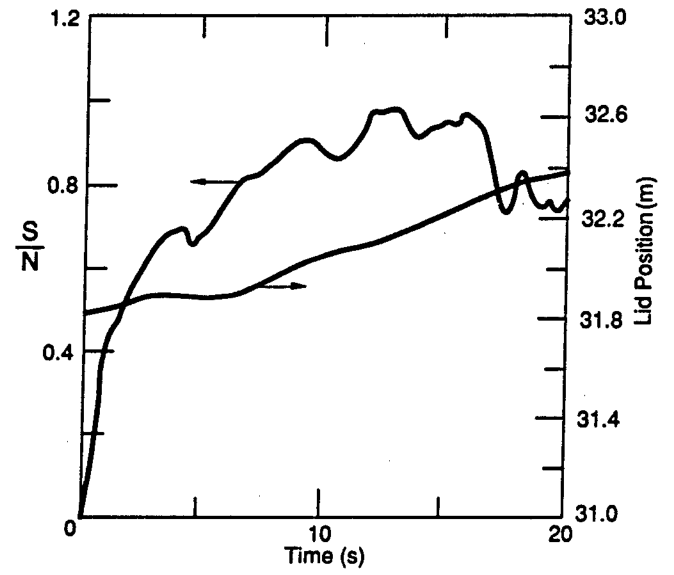


Figure 16. Typical time history of S/N *and lid displacement.*

the shear box and the vertical displacement of the lid were recorded at constant intervals during the experiments. The experimental and material parameters used were:

Box width	45 m
Box depth	30 m (variable)
U_{shear}	$0.25\ \mathrm{m\ s^{-1}}$
N	1.34, 5.34, 21.36 kPa
Rubble block thickness	0.5 m
k_{ne}	$1.25 \times 10^6, 5.0 \times 10^6, 20.0 \times 10^6\ \mathrm{N\ m^{-2}}$
k_{nv}	$2\ C_d\ mass(k_{ns}/mass)^{0.5}$
C_d	0.9
k_{te}	$10^6\ \mathrm{N\ m^{-1}}$
μ	0.1, 0.5, 1.0
ρ_{ice}	$920\ \mathrm{kg\ m^{-3}}$
σ_{cr}	350 kPa.

Figure 16 shows a typical time history of the shear-to-normal-force ratio S/N and the lid displacement in the direct shear experiment. Graphs of S/N showed a steady rise to an initial peak. The lid displacement ranged from a few centimeters for $\mu = 0.1$ to a meter or more for $\mu = 1.0$. The large lid displacements were a result of changes in the material porosity in the vicinity of the nominal failure surface and the presence of voids created at the leading ends of the shear box. At high loads the lid displacement was reduced by the effects of breakage.

Figure 17 shows the ratio of maximum shear force to applied load S/N versus applied pressure for three values of the friction coefficient μ. The results show that the shear strength depends strongly on friction and load. The dependence of S/N on friction was expected. The dependence on load is less obvious. The shear force S is the

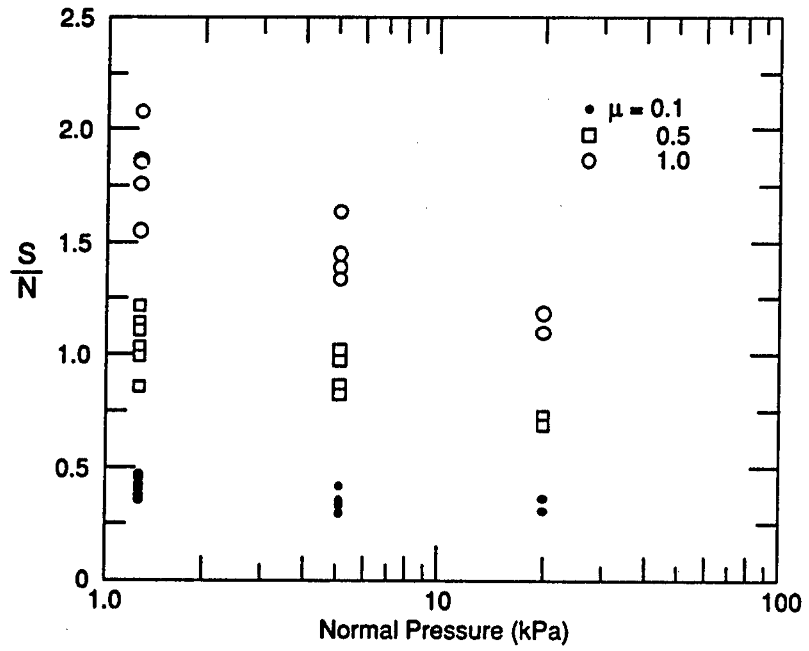


Figure 17. Shear to normal force S/N *versus applied pressure.*

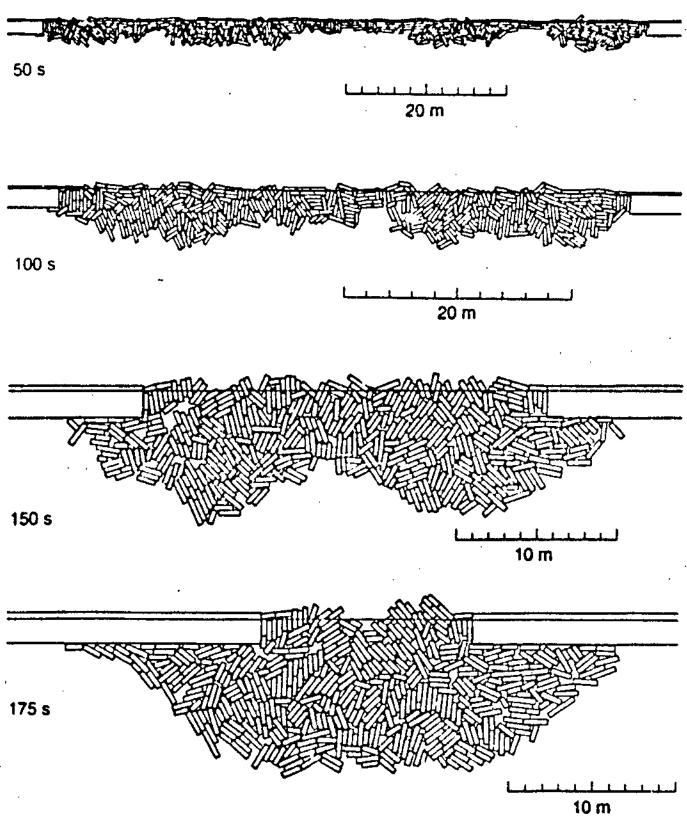


Figure 18. Simulation of a pressure ridge grown from a rubble-filled lead.

sum of many local forces on the nominal failure plane. Because of the enforced shear along the failure plane, these local forces will increase until they are relieved by local rearrangement or breakage. Breakage is a function of the magnitude of the local forces, which depend primarily on the load and secondarily on friction and shape. If breakage occurs, the rearrangement will be terminated and the local forces in the rubble will not reach the level that they would have reached during rearrangement. Therefore, breakage will increase with load and will manifest itself as a reduction in the frictional component of the shear strength. To demonstrate the effect of breakage on shear strength, the experiments were repeated without breakage. The dependence of S/N on the applied load was drastically reduced.

4.2 Model of ridging in leads filled with ice rubble

Hopkins et al. (1991) describe a two-dimensional simulation of sea-ice ridges formed from a floating layer of rubble compressed between converging multi-year floes. The simulations began with two ice floes of equal thickness separated by a lead covered by a single thickness of rubble blocks. As the opposing floes converged, the layer of rubble thickened as blocks were thrust over and under neighboring blocks. As the layer of rubble thickened, the resisting force on the two thick floes increased. The ice floes themselves were composed of a rectangular array of square blocks. The blocks making up each floe moved as a unit in the horizontal direction at a constant velocity.

Figure 18 shows a sequence of snapshots from a typical ridging simulation. The scale is readjusted in each snapshot. The experimental and material parameters used

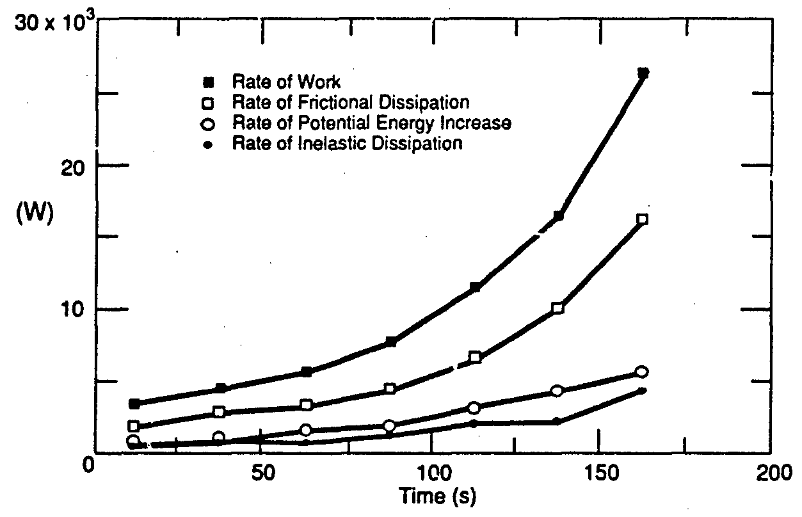


Figure 19. Energetics of the simulated ridge growth shown in Figure 18.

in the experiment were:

Initial lead width	100 m
Floe thickness	1.8 m
Floe speed	0.25 m s^{-1}
Number of blocks	450
Block thickness	0.308 m
Average block length	1.1 m
ρ_i	920.0 kg m^{-3}
ρ_w	1010.0 kg m^{-3}
σ_{cr}	350 kPa
k_{ne}	4.0×10^6 N m^{-2}
k_{nv}	62.4×10^3 N m^{-1} s ($C_d = 0.9$)
k_{te}	1.0×10^6 N m^{-1}
μ	0.5.

Figure 19 illustrates the overall rate of work and its three components for the ridge growth shown in Figure 18. The data points represent average values over a 25-second interval. The rate of increase of potential energy is calculated from eq 28 and the rates of frictional and inelastic dissipation are calculated from eq 29 and 30. The overall rate of work is the product of the floe velocities and the resisting force on the floes integrated over each 25-second interval of the experiment.

A series of experiments was conducted to establish the dependence of the energy consumed in ridging on the velocity of the multi-year floes and on the roughness and the inelasticity of the rubble blocks. The experiments show that the ice roughness, characterized by the friction coefficient μ, is the most significant factor in determining the energetics of ridging.

While instances of ridges grown from rubble-filled leads probably do occur in the central Arctic pack, this model seems better suited to areas of the pack in the vicinity of land. A more likely model for ridge growth in the central pack is for a sheet of thin ice covering a newly frozen lead to be driven against a multi-year floe by the converging pack.

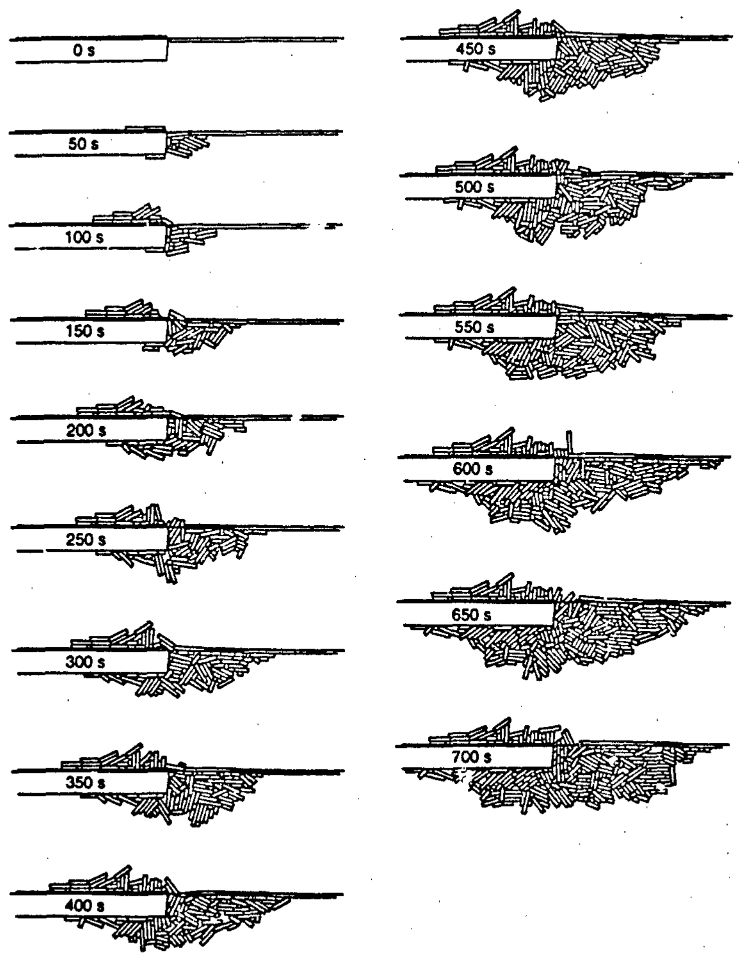


Figure 20. Simulation of the ridging of a sheet of unbroken ice covering a lead.

4.3 Modeling ridging of a lead covered with a sheet of thin ice

Hopkins and Hibler (1991b) describe a two-dimensional simulation of ridges formed from an unbroken sheet of newly frozen lead ice colliding with a thick multi-year floe. Snapshots of the simulated ridge growth are shown in Figure 20. Blocks broken from the leading edge of the thin ice sheet collect above and beneath the multi-year floe to form the characteristic ridge structure seen in the central Arctic. The results of preliminary numerical experiments using this model indicate that the amount of energy required to ridge ice may be much larger than previous estimates.

The ridging simulation begins with a thin plate of ice moving at a constant speed toward a thick multi-year floe. As the lead ice collides with the floe, it bends. The lead ice is modeled as a plate on an elastic foundation, using standard solutions (Hetenyi 1946) for vertical and angular deflection and moment at regular points in the plate. (Future experiments will use the beam simulation described in section 3.3). When the moment at a point in the lead ice exceeds its flexural strength, it breaks at that point. As the lead ice continues to move toward the floe, additional blocks break off and contribute to the ridge structure. Occasionally, the blocks themselves fail in flexure. The energy dissipated at every contact by friction and inelasticity and the changes in potential energy of the rubble blocks and the thin ice plate are calculated using eq 28–

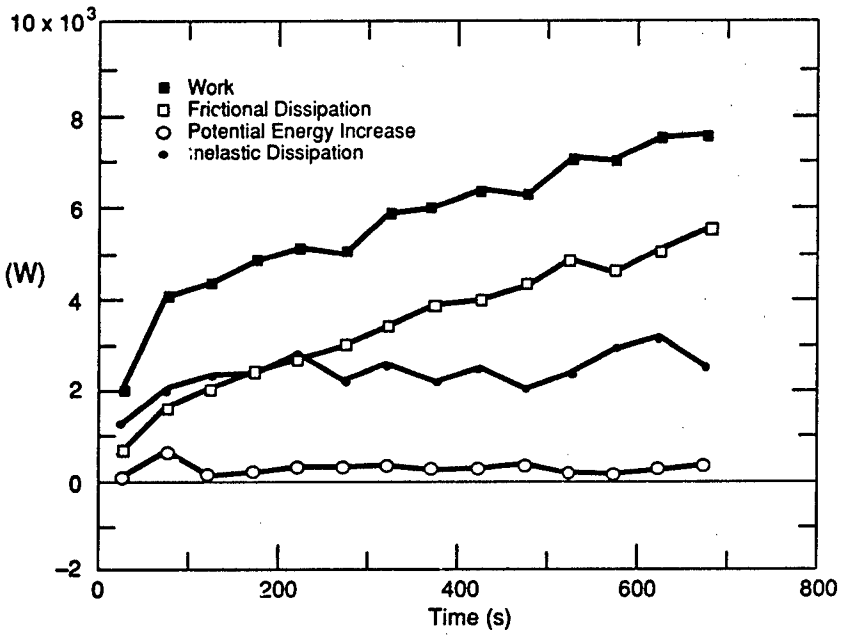


Figure 21. Energetics of the simulated ridge growth shown in Figure 20.

30. The overall ridging work is the product of the velocity of the lead ice and the resisting force on the lead ice integrated over each interval. The multi-year floe is treated as a rigid plate. The experimental and material parameters used in the simulation were:

Floe thickness	2.0 m
Lead ice thickness	0.3 m
Speed of lead ice	0.33 m s^{-1}
ρ_i	920.0 kg m^{-3}
ρ_w	1010.0 kg m^{-3}
σ_{cr}	300 kPa
k_{ne}	4×10^6 N m^{-2}
k_{nv}	4×10^3 N $(m/s)^{-1}$
k_{te}	1×10^6 N m^{-2}
μ dry	0.40
μ wet	0.25
Young's modulus	1.0×10^7 N m^{-2}.

The rates of work, dissipation and increase of potential energy averaged over 50-second intervals are shown in Figure 21. The overall rate of work increases in an approximately linear fashion throughout the experiment, while the rate of increase of potential energy remains relatively constant. The overall rate of work is largely a function of the frictional force exerted on the underside of the lead ice. The frictional force, which depends on the buoyancy of the rubble, is proportional to the area of the rubble. Since the lead ice is fed into the ridge at a constant rate, the area of rubble beneath the plate, the frictional force and the overall work will increase linearly with time. The ratio of the overall rate of work to the rate of increase of potential energy in Figure 21 varies between 10:1 and 20:1. In the ridges grown from a rubble-filled lead discussed in section 4.2, the ratio was between 3:1 and 5:1, depending on the coefficient of friction.

The force required to ridge ice largely determines the internal strength of the ice pack in compression. In large-scale modeling of the ice covering the Arctic Basin, using an ice thickness distribution, it is convenient to parameterize total ridging work in terms of the increase of potential energy from ridging. The results of the ridging simulation suggest that the value of two, which has been typically assumed, may greatly underestimate the strength of the ice pack.

5. CONCLUSION

This work has outlined, in detail, the component mechanisms that together make up a method for the simulation of two-dimensional systems of discrete, convex polygonal particles. The technique has been used to do numerical shearbox experiments (section 4.1) (Hopkins and Hibler 1991a), to simulate pressure ridging of a layer of floating ice rubble (section 4.2) (Hopkins et al. 1991), and to simulate pressure ridging of an intact layer of thin ice (section 4.3) (Hopkins and Hibler 1991b).

The present work owes a debt to the DIBS model (Walton 1980). In the spirit that a student may only repay his teacher by becoming better, the present work contains several improvements.

1. In the DIBS model, certain types of intersections between polygons were problematical and were handled as exceptions. The strategy of O'Rourke et al. (1982) (section 2.4.1) used in the present work has removed these problems.

2. The present work uses an Elastic–Viscous–Plastic (EVP) contact force model (section 2.4.3) based on the area of intersection of polygons in contact. Walton outlined a similar model for the elastic force, but used velocity-dependent damping in parallel. This damping method is unappealing in point on point and point on side contacts because the damping force is a maximum when the area of intersection is a minimum, namely, when the point first contacts the other particle. The logical counterpart to an elastic force proportional to the area of intersection is a viscous force proportional to the rate of change of the area. However, this damping method does not dissipate enough energy in point on point and point on side contacts to model highly inelastic behavior. The essential ingredient in this scheme is the plastic limit on the stress across the plane of contact, which qualitatively accounts for the initial crushing that occurs in point on point and point on side contacts.

3. A failure criterion suitable for slender rigid blocks has been developed and implemented (section 2.7).

4. A "glued" joint algorithm suitable for dynamic simulation of specimens composed of discrete polygonal elements has been developed. This approach is consistent with the basic polygonal particle simulation. The strength of the glued joints depends on the material properties of the modeled material. A viscous–elastic force model is used to calculate the forces between adjacent blocks. If the stresses in the fibers at the end of a joint exceed the tensile strength of the material, a crack is initiated, which propagates across the joint at a constant speed. The algorithm permits two-dimensional modeling of situations in which failure of an intact material leads to creation of rubble, such as in Arctic ice ridging and river ice jamming.

5. The computer code itself has been written with clarity and relative simplicity. It represents a huge improvement over previous codes. This is not cosmetic. If a code is to be a useful tool, it must be capable of being quickly shaped and adapted to perform new tasks.

6. LITERATURE CITED

Abbott, E.A. (1884) *Flatland*. New York: Dover, 6th ed.
Cundall, P. (1971) A computer model for simulating progressive large scale movements in blocky rock systems. In *Proceedings of the Symposium on Rock Fracture, Nancy, France*. International Society of Rock Mechanics, paper II-8.
Cundall, P.A. (1974) Rational design of tunnel supports: A computer model for rock mass behavior using interactive graphics for the input and output of geometrical data. Washington D.C.: U.S. Army Corps of Engineers, Technical Report MRD-2-74.
Cundall, P.A. (1980) UDEC—A generalised distinct element program for modelling jointed rock. Defense Technical Information Center, DTIC No. PCAR-1-80.
Cundall, P., J. Marti, P. Beresford, N. Last, M. Asgain and T. Maini (1978) Computer modeling of jointed rock masses. Los Angeles: Dames and Moore, Technical Report N-78-4.
Hetenyi, M. (1946) *Beams on Elastic Foundations*. Ann Arbor: University of Michigan Press.
Hopkins, M.A. (1991) A particle beam simulation. In *Mechanics Computing in 1990's and Beyond, Proceedings of the Engineering Mechanics Division, ASCE, 20-22 May, Columbus, Ohio*, p. 1274–1278.
Hopkins, M.A. and W.D. Hibler III (1991a) On the shear strength of geophysical scale ice rubble. *Cold Regions Science and Technology*, **19**: 201–212.
Hopkins, M.A. and W.D. Hibler III (1991b) On the ridging of a thin sheet of sea ice. *Annals of Glaciology*, **15**: 81–86.
Hopkins, M.A., W.D. Hibler III and G.M. Flato (1991) On the numerical simulation of the sea ice ridging process. *Journal of Geophysical Research*, **96**: 4809–4820.
O'Rourke, J., C.B. Chien, T. Olson and D. Naddor (1982) A new linear algorithm for intersecting convex polygons. *Computer Graphics and Image Processing*, **19**: 384–391.
Parmerter, R.R. and M.D. Coon (1972) Model of pressure ridge formation in sea ice. *Journal of Geophysical Research*, **77**: 6565–6575.
Potapov, A.V., M.A. Hopkins and C.S. Campbell (in prep.) A two-dimensional dynamic simulation for particle fracture. Part 1: Description of the model.
Preparata, F.P. and M.I. Shamos (1985) *Computational Geometry: An Introduction*. New York: Springer-Verlag.
Timoshenko, S. (1955) *Vibration Problems in Engineering*. New York: Van Nostrand, 3rd ed.
Walton, O.R. (1980) Particle dynamics modeling of geological materials. Livermore, California: University of California, Lawrence Livermore Laboratory, UCRL 52915.

APPENDIX A: A GENERAL TWO-DIMENSIONAL POLYGONAL PROGRAM

The following program contains the general two-dimensional simulation technique described above. Although the program is written in Fortran 8x, it might be more accurately described as being written in a Pascal style. That is, all variables are declared explicitly and all variables which are not local to a subroutine are global, or common, to all subroutines. The common blocks of global variables and their declarations are contained in the 'include' file blockcom.f.

List of program variables

Np:	the maximum number of blocks.
Npr:	the maximum number of neighbors per block.
NpNpr:	the maximum number of pairs of blocks.
Nvmax:	the maximum number of vertices per block.
dimi,dimj:	the maximum number of grid cells in the x and y directions.
dimk:	the maximum number of blocks per grid cell.
dimij:	the maximum number of grid cells.
dimijk:	the maximum number of spaces in the 3-D grid array.

List of Program Variables:

x,y,th:	1-D arrays, store position (x,y) and orientation (theta) of each block at time n.
u,v,w:	1-D arrays, store velocities of each block at time n-1/2 (w=rotational velocity).
Mass,Jpol:	1-D arrays, store mass and polar moment of inertia of each block.
r,a:	2-D arrays, store length and angle of vectors from center to vertices.
rcosa,rsina:	2-D arrays, store r*sin(a) and r*cos(a) for vectors from center to vertices.
rx,ry:	2-D arrays, store x and y position of vertices with respect to centers.
length:	2-D array, stores the length of each side of each polygon.
vecmidn,vecmidt:	2-D array, stores the normal and tangential components of the vector from the polygon center to the midpoint of each side of each polygon.
home,near:	1-D arrays, store numbers of home and near polygons in a pair of near neighbors.
nnear:	1-D array, stores the number of near neighbors of each polygon.
grid:	3-D array, corresponds to the grid overlaying the simulation domain. Used to define the neighborhood surrounding the polygons. The third dimension stores the array numbers of blocks whose centers lie in the given cell.
ngrid:	2-D array, stores the number of blocks whose centers lie in each cell of the 2-D grid.
invx,invy:	1-D arrays, stores the column and row location of the polygons in the grid.

ptli,ptlj:	1-D arrays, used to initialize the current vertices in the overlap subroutine.
Ftp:	1-D array, stores the frictional force between a pair of polygons at time n-1.
vnormp,vtangp:	1-D arrays, store the normal and tangential relative velocities between a pair of polygons at time n-1.
Areap,dAreap,ElAreap:	1-D arrays, store the total area of overlap, the change in the total area, and the area of elastic deformation of a contact at time n-1.
slopen,intern:	1-D arrays, store information about the deformation of each glued joint in the normal direction.
slopet,intert:	1-D arrays, store information about the deformation of each glued joint in the tangential direction.
dslopen,dintern:	1-D arrays, store information about the rate of change of the deformation of each glued joint in the normal direction.
dslopet,dintert:	1-D arrays, store information about the rate of change of deformation of each glued joint in the tangential direction.
crackt,crackb:	1-D arrays, stores the length of cracks from the top and bottom ends of each glued joint.
vh1,vh2,vn1,vn2:	1-D arrays, store vertices of the home and near polygons bounding each glued joint.
status:	1-D arrays, stores the status of the joint between a pair of blocks. That is, intact (glued) or broken.
F1,F2,M:	1-D arrays, store the x and y components of force and the torque on a polygon at time n.
F2b,Mb:	1-D arrays, store force and torque on a polygon at time n due to gravity and buoyancy.
kne:	contact stiffness per meter of contact width in the normal direction.
knv:	contact viscosity per meter of contact width in the normal direction.
kte:	contact stiffness in the tangential direction.
ktv:	contact viscosity in the tangential direction.
mu:	contact friction coefficient in the tangential direction.
rhoi,rhow:	ice and water density.
csigmac:	compressive strength of the block material.
gsigmat:	tensile strength of a glued joint.
grav:	acceleration of gravity.
cell:	edge dimension of each square grid cell.
width,height:	width and height of the simulation domain.
work,idiss,fdiss,penrg:	store total work on system, inelastic dissipation, frictional dissipation, and change of potential energy.
epsilon:	maximum separation between close contacts.
dt:	simulation time step.
time0,time,duration:	initial time, current time, and experiment duration.
tflt:	time interval between calls to bouyancy subroutine.
tout:	time interval between calls to output subroutine.
tpic:	time interval between calls to graphics subroutine.
N,Nbnd,Nt:	number of boundary and non-boundary polygons and total number of polygons.

nsearch:	interval between calls to global search subroutine (number of time steps).
cfile0:	name of the initial and final configuration and the output.
cfile:	1-D array, stores names of the configuration files dumped at intervals for restart.
rfile:	1-D array, stores names of the result files dumped at intervals.

Program listing

```
      program A_Block

C Two-dimensional, Irregular polygonal blocks
C Glued-joint contacts and collisional contacts
C Elastic-viscous-plastic contact force model

      include 'blockcom.f'
      external initvars

      Call Initial

      Do While (time-time0 .lt. duration)

        if (tpict .ge. tpic) Call Picture

C skip the search block for nsearch time steps

        if (scount .ge. nsearch) then
          Call Gridfill
          Call Search

          scount = 0
        end if

        if (tfltt .ge. tflt) Call Float

        Call Force
        Call MoveBlocks

        if (toutt .ge. tout) Call Output

        scount = scount+1

      End Do

      End
*****************************************************************

C Blockcom.f contains all common blocks and Global declarations.
C This version is for the new glued joint simulation.
C
      Implicit None
      Integer*4 Np,Npr,NpNpr,Nvmax,dimi,dimj,dimk,dimij,dimijk
      Parameter (Np=600, Npr=10, NpNpr=6000, Nvmax=10, dimi=200,
     &           dimj=30, dimk=15, dimij=6000, dimijk=90000)

C Grid and inverse address variables:

      Common /GridVar1/ grid(dimi,dimj,dimk),ngrid(dimi,dimj),
     &           invx(Np),invy(Np)

      Integer*2 grid,ngrid,invx,invy

C Particle configuration and shape variables:

      Common /BlocVar1/ x(Np),y(Np),th(Np)
      Common /BlocVar2/ u(Np),v(Np),w(Np)
      Common /BlocVar3/ mass(Np),Jpol(Np),Asub(Np)
      Common /BlocVar4/ r(Np,Nvmax),a(Np,Nvmax),rcosa(Np,Nvmax),
     &           rsina(Np,Nvmax),rx(Np,Nvmax),ry(Np,Nvmax),
      Common /BlocVar5/ length(Np,Nvmax),vecmidn(Np,Nvmax),
     &           vecmidt(Np,Nvmax)
      Common /BlocVar6/ Nv(Np)

      Real*8 x,y,th
      Real*8 u,v,w
      Real*8 Mass,Jpol,Asub
      Real*8 r,a,rcosa,rsina,rx,ry
      Real*8 length,vecmidn,vecmidt
      Integer*2 Nv

C Pair definition variables:

      Common /PairVar1/ near(NpNpr),nnear(Np)
      Common /PairVar2/ status(NpNpr)

      Integer*2 near,nnear
      Character*1 status

C Glued contact variables:

      Common /GlueVar1/ vh1(NpNpr),vh2(NpNpr),vn1(NpNpr),vn2(NpNpr)
      Common /GlueVar2/ slopen(NpNpr),dslopen(NpNpr),slopet(NpNpr),
     &           dslopet(NpNpr),intern(NpNpr),dintern(NpNpr),
     &           intert(NpNpr),dintert(NpNpr)
      Common /GlueVar3/ crackt(NpNpr),crackb(NpNpr)

      Integer*2 vh1,vh2,vn1,vn2
      Real*8 siopen,dslopen,slopet,dslopet,intern,dintern,intert,dintert
      Real*8 crackt,crackb

C Collisional or non-glued contact variables:

      Common /CollVar1/ pt1i(NpNpr),pt1j(NpNpr)
      Common /CollVar2/ Ftp(NpNpr),vnormp(NpNpr),vtangp(NpNpr),
     &           Areap(NpNpr),dAreap(NpNpr),ElAreap(NpNpr)

      Integer*2 pt1i,pt1j
      Real*8 Ftp,vnormp,vtangp,Areap,dAreap,ElAreap
```

```
C Material parameters:

        Common /MatVar1/ kne,kte,knv,ktv,mu
        Common /MatVar2/ csigmac,gsigmat,vcrack
        Common /MatVar3/ rhoi,rhow,tlead,edge

        Real*8 kne,kte,knv,ktv,mu
        Real*8 csigmac,gsigmat,vcrack
        Real*8 rhoi,rhow,tlead,edge

C Program variables:

        Common /ForceVar/ F1(Np),F2(Np),M(Np),F2b(Np),Mb(Np)
        Common /ParamVar/ pi,nul,epsilon,cell,height,width,grav,
     &                waterlevel
        Common /RsultVar/ work,wwork,idiss,fdiss,penrg,KEini
        Common /TimerVar/ dt,time0,time,duration,tpict,tpic,toutt,tout,
     &                tfltt,tflt
        Common /CountVar/ N,Nbnd,Nt,nsearch,scount,count,fcount,nfiles
        Common /Filevar/ cfile0,cfile(30),rfile(30)

        Real*8 F1,F2,M,F2b,Mb
        Real*8 pi,nul,epsilon,cell,height,width,grav,waterlevel
        Real*8 work,wwork,idiss,fdiss,penrg,KEini
        Real*8 dt,time0,time,duration,tpict,tpic,toutt,tout,tfltt,tflt
        Integer*2 N,Nbnd,Nt,nsearch,scount,count,fcount,nfiles
        Character*12 cfile0,cfile,rfile

****************************************************************

        Subroutine CollForce(ind,nh,nn,nx,ny,Area,xcen,ycen,base,
     &                F1h,F2h,Mh,Mn,ndiss,tdiss)

C The CollForce subroutine calculates the interparticle force.

C A general contact force algorithm is used. First, all points on one


C polygon interior to the other are found. Second, The points of
C intersection are found. Third, the area of overlap and centroid are
C calculated.

C The force has normal and tangential components. The normal force
C has an elastic component proportional to the area of overlap
C and a viscous component proportional to the rate of change of
C the area of overlap. The magnitude of the normal force is not
C allowed to exceed a limit value equal to the product of the
C compressive strength of the material and the breadth of the
C contact surface. The tangential force is incrementally increasing.
C The increments are proportional to the tangential velocity at the
C point of contact. The tangential force must be less than mu times
C the normal force. See section 2.4.3 of the report.

C the total area of deformation is stored in Areap(i)
C the area of elastic deformation is stored in ElAreap(i)

        include 'blockcom.f'

        Integer*2 ind,nh,nn,k
        Real*8 nx,ny,Area,xcen,ycen,base
        Real*8 F1h,F2h,Mh,Mn,ndiss,tdiss
```

```
      Real*8 Fn,Fne,Fnv,Ft,Fpl,vnorm,vtang
      Real*8 armhx,armhy,armnx,armny,dx,dy,du,dv
      Real*8 dArea,ElArea,dArean,dAreanmh

      dArea = Area-Areap(ind)
      Areap(ind) = Area

      If (Area .gt. 0.0 .and. ElAreap(ind)+dArea .gt. 0.0) then

C Moment armh of the area of overlap about the polygon centers of mass:

        armhx = xcen-x(nh)
        armhy = ycen-y(nh)
        armnx = xcen-x(nn)
        armny = ycen-y(nn)

C Relative velocity at centroid of overlap area:

        du = u(nh)-u(nn)-armhy*w(nh)+armny*w(nn)
        dv = v(nh)-v(nn)+armhx*w(nh)-armnx*w(nn)

C Relative velocity at time n-1/2:

        vnorm = nx*du+ny*dv
        vtang = nx*dv-ny*du

C The normal component of force

        dAreanmh = dArea/dt

        if (dAreap(ind) .ne. 0.0) then
          dArean = 1.5*dAreanmh-0.5*dAreap(ind)
        else
          dArean = 0.0
        end if

C The elastic component of the normal force

        Fne = kne*(ElAreap(ind)+dArea)

C The viscous component of the normal force

        Fnv = knv*dArean

C The plastic limit on the normal force

        Fpl = csigmac*base

C The total normal force (must be > zero)

        Fn = max(nul,Fne+Fnv)

        if (Fn .lt. Fpl) then
          ElArea = ElAreap(ind)+dArea
        else
          Fn = Fpl

C This is a solution of equation (51).  The solution must be
C used with care if knv is relatively small.
```

```
               if (knv.gt.0.0)

            then ElArea = (Fpl+(0.5*knv/dt)*
     &                   (3.0*ElAreap(ind)+dt*dAreap(ind)))/(kne+1.5*knv/dt)
               else
                  ElArea = ElAreap(ind)
               end if
            end if

C The tangential frictional force (coulomb limit = mu*Fn)

            Ft = Ftp(ind)-kte*dt*vtang
            Ft = sign(min(abs(Ft),mu*Fn),Ft)

C The x and y components of the force on the HOME polygon

            F1h = nx*Fn-ny*Ft
            F2h = ny*Fn+nx*Ft

C The moments of the forces on the HOME and NEAR polygons

            Mh = armhx*F2h-armhy*F1h
            Mn = armny*F1h-armnx*F2h

C Inelastic energy dissipation

            if (vnormp(ind) .ne. 0.0) then
               ndiss = -Fn*(1.5*vnorm-0.5*vnormp(ind))
            else
               ndiss = -Fn*vnorm
            end if

C Frictional energy dissipation

            if (vtangp(ind) .ne. 0.0) then
               tdiss = -Ft*(1.5*vtang-0.5*vtangp(ind))
            else
               tdiss = -Ft*vtang
            end if

C Save necessary variables for the next time step

            Ftp(ind) = Ft
            vnormp(ind) = vnorm
            vtangp(ind) = vtang
            if (ElAreap(ind) .gt. 0.0)
     &                  dAreap(ind) = (ElArea-ElAreap(ind))/dt
            ElAreap(ind) = ElArea

         Else
            F1h = 0.0
            F2h = 0.0
            Mh = 0.0
            Mn = 0.0
            tdiss = 0.0
            ndiss = 0.0

            Ftp(ind) = 0.0
            vnormp(ind) = 0.0
            vtangp(ind) = 0.0
```

```
      dAreap(ind) = 0.0
      ElAreap(ind) = 0.0
    End If

    End

************************************************************

      Subroutine Contact(nh,nn,neighbor)

C A brute force routine to find the shortest distance between two
C polygons. The routine quits when it finds a distance less than
C epsilon. It is used to reduce the number of pairs in the list
C of near neighbors.

      include 'blockcom.f'

      Real*8 dx,dy,maxsep,xprod,ssx(Nvmax),ssy(Nvmax)
      Integer*2 j,jp,nh,nn,pt
      Logical neighbor

C Make side vectors - NEAR block:

      Do j = 1,Nv(nn)

        jp = 1+mod(j,Nv(nn))

        ssx(j) = (rx(nn,jp)-rx(nn,j))/length(nn,j)
        ssy(i) = (ry(nn,jp)-ry(nn,j))/length(nn,j)

      End Do

C Find vertex of HOME block closest to NEAR block:

      pt = 1

      Do While (.not. neighbor .and. pt .le. Nv(nh))

        dx = x(nh)-x(nn)+rx(nh,pt)
        dy = y(nh)-y(nn)+ry(nh,pt)

        maxsep = 1000.0

        Do j = 1,Nv(nn)

          xprod = ssx(j)*(dy-ry(nn,j))-ssy(j)*(dx-rx(nn,j))

          if (xprod .lt. maxsep) maxsep = xprod

        End Do

        if (maxsep .ge. -epsilon) neighbor = .true.

        pt = pt+1

      End Do

      If (.not. neighbor) then

C Make side vectors - HOME block:
      Do j = 1,Nv(nh)

        jp = 1+mod(j,Nv(nh))

        ssx(j) = (rx(nh,jp)-rx(nh,j))/length(nh,j)
        ssy(j) = (ry(nh,jp)-ry(nh,j))/length(nh,j)

      End Do

C Find vertex of NEAR block closest to HOME block:

      pt = 1

      Do While (.not. neighbor .and. pt .le. Nv(nn))

        dx = x(nn)-x(nh)+rx(nn,pt)
        dy = y(nn)-y(nh)+ry(nn,pt)

        Do j = 1,Nv(nh)

          xprod = ssx(j)*(dy-ry(nh,j))-ssy(j)*(dx-rx(nh,j))

          if (xprod .lt. maxsep) maxsep = xprod

        End Do

        if (maxsep .ge. -epsilon) neighbor = .true.

        pt = pt+1

      End Do

    End If

    End
*****************************************************************

      Subroutine Float

C This subroutine calculates the buoyant force on each block.
C The submerged area is the area of overlap with the water polygon.
C The water polygon is defined in the initial subroutine.

      include 'blockcom.f'

      Integer*2 i,j,ind
      Real*8 nx,ny,area,xcen,ycen,base

      j = Np
      ind = NpNpr

      Do i = Nbnd+1,Nt

        Call Overlap(ind,j,i,nx,ny,area,xcen,ycen,base)

        if (area .eq. 0.0 .and. y(i) .lt. waterlevel) then
          area = mass(i)/rhoi
          xcen = x(i)
        end if
```

```
       F2b(i) = grav*(mass(i)-rhow*area)
       Mb(i) = -grav*rhow*area*(xcen-x(i))

     End Do

     tfltt = 0.0

     End

*********************************************************************

      Subroutine Force

C The Force subroutine calls the collisional contact or the glued
C contact force subroutine depending on the status of the joint.

      include 'blockcom.f'

      Real*8 F1h,F2h,Mh,Mn,ndiss,tdiss
      Real*8 nx,ny,area,xcen,ycen,base
      Integer*2 ind,j,nh,nn

      Do nh = Nbnd+1,Nt

        ind = Npr*(nh-1)

        Do j = 1,nnear(nh)

          ind = ind+1
          nn = near(ind)

          If (status(ind) .eq. 'i') then

            Subroutine GlueForce(ind,nh,nn,F1h,F2h,Mh,Mn,ndiss,tdiss)

          Else


            Subroutine Overlap(ind,nh,nn,nx,ny,area,xcen,ycen,base)

            Subroutine CollForce(ind,nh,nn,nx,ny,area,xcen,ycen,base,
     &                   F1h,F2h,Mh,Mn,ndiss,tdiss)

          End If

C Add forces for each contact to find the resultant force on each polygon.

          F1(nh) = F1(nh)+F1h
          F2(nh) = F2(nh)+F2h
          M(nh) = M(nh)+Mh

          F1(nn) = F1(nn)-F1h
          F2(nn) = F2(nn)-F2h
          M(nn) = M(nn)+Mn

          idiss = idiss+ndiss
          fdiss = fdiss+tdiss

        End Do
        End Do

        End
*****************************************************************

        Subroutine GlueForce(i,nh,nn,F1h,F2h,Mh,Mn,ndiss,tdiss)

        include 'blockcom.f'

        Real*8 x1,x2,x3,x4,y1,y2,y3,y4,deltax1,deltay1,deltax2,deltay2
        Real*8 slopenn,dslopenn,dslopennmh,slopetn,dslopetn,dslopetnmh
        Real*8 internn,dinternn,dinternnmh,intertn,dintertn,dintertnmh
        Real*8 Fn,Fne,Fnv,Fte,Ft,Ftv,Ftmh,MFne,MFnv,MFt
        Real*8 dx,dy,dxn,dyn,dify1,dify2,dify3,sig1,sig2,half,tlength
        Integer*2 i,nn,nh

        tlength = length(nh,vh2(i))
        half = 0.5*tlength

C vh1 and vh2 are the numbers of the vertices on block nh.
C vn1 and vn2 are the numbers of the vertices on block nn.

C Find the coordinates of the top and bottom pairs of vertices in space:

        x1 = x(nh)+rx(nh,vh1(i))
        y1 = y(nh)+ry(nh,vh1(i))
        x2 = x(nh)+rx(nh,vh2(i))
        y2 = y(nh)+ry(nh,vh2(i))

        x3 = x(nn)+rx(nn,vn1(i))
        y3 = y(nn)+ry(nn,vn1(i))
        x4 = x(nn)+rx(nn,vn2(i))
        y4 = y(nn)+ry(nn,vn2(i))

C A local x,y coordinate frame is used in the subroutine which
C corresponds to the η,ζ notation used in section 3.1.
C The normal (x) direction is perpendicular to the face of the
C blocks and the tangential (y) direction is tangent to the
C face of the blocks. The overall flow coordinate frame is also
C designated x,y. The components of the tangent unit vector in
C the flow frame are dx,dy. The components of the normal unit
C vector in the flow frame are then dy,-dx.

        dx = (x1-x2)/tlength
        dy = (y1-y2)/tlength

C The array variable crackb stores the length of a crack which was
C initiated at the negative end of the joint. Similarly, the array
C variable crackt stores the length of a crack which was initiated
C at the positive end of the joint.

C If there is a crack at the negative end, recalculate the positions
C of the points on blocks nh and nn at the crack tip:

        if (crackb(i) .gt. 0.0) then
          dxn = (x3-x4)/tlength
          dyn = (y3-y4)/tlength
          x2 = x2+crackb(i)*dx
          y2 = y2+crackb(i)*dy

          x4 = x4+crackb(i)*dxn
          y4 = y4+crackb(i)*dyn
        end if

C If there is a crack at the positive end, recalculate the positions
C of the points on blocks nh and nn at the crack tip:

        if (crackt(i) .gt. 0.0) then
          dxn = (x3-x4)/tlength
          dyn = (y3-y4)/tlength

          x1 = x1-crackt(i)*dx
          y1 = y1-crackt(i)*dy

          x3 = x3-crackt(i)*dxn
          y3 = y3-crackt(i)*dyn
        end if

C Calculate the fiber deformation at positive end (1) and the
C negative (2) end (or at crack tips):

        deltax1 = (x3-x1)*dy-(y3-y1)*dx
        deltay1 = (x3-x1)*dx+(y3-y1)*dy
        deltax2 = (x4-x2)*dy-(y4-y2)*dx
        deltay2 = (x4-x2)*dx+(y4-y2)*dy

C Calculate the y position and the stress in the fibers at
C the positive and negative ends:

        y1 = half-crackt(i)
        y2 = -half+crackb(i)

        sig1 = kne*deltax1
        sig2 = kne*deltax2

C Tensile fracture at the positive end of joint
C (tensile stress is positive):

        if (sig1 .ge. gsigmat) crackt(i) = crackt(i)+dt*vcrack

C Tensile fracture at the negative end of joint:

        if (sig2 .ge. gsigmat) crackb(i) = crackb(i)+dt*vcrack

C Use difference variables for efficiency:

        dify1  y1-y2
        dify2  (y1*y1-y2*y2)/2.0
        dify3  (y1**3-y2**3)/3.0

C Calculate the slope of deltax and the time derivative
C of the slope at time n:

        slopenn = (deltax1-deltax2)/dify1

        if (slopen(i) .ne. 0.0) then
          dslopennmh = (slopenn-slopen(i))/dt
          else
            dslopennmh = 0.0
          end if

          dslopenn = 1.5*dslopennmh-0.5*dslopen(i)

C Calculate the intercept of deltax and the time derivative
C of the intercept at time n:

          internn = (y1*deltax2-y2*deltax1)/dify1

          if (intern(i) .ne. 0.0) then
            dinternnmh = (internn-intern(i))/dt
          else
            dinternnmh = 0.0
          end if

          dinternn = 1.5*dinternnmh-0.5*dintern(i)

C Elastic component of the normal force and its moment at time n:

          Fne = kne*(slopenn*dify2+internn*dify1)
          MFne = -kne*(slopenn*dify3+internn*dify2)

C Viscous component of the normal force and its moment at time n:

          Fnv = knv*(dslopenn*dify2+dinternn*dify1)
          MFnv = -knv*(dslopenn*dify3+dinternn*dify2)

C Slope of deltay and the time derivative of the slope at time n:

          slopetn = (deltay1-deltay2)/dify1

          if (slopet(i) .ne. 0.0) then
            dslopetnmh = (slopetn-slopet(i))/dt
          else
            dslopetnmh = 0.0
          end if

          dslopetn = 1.5*dslopetnmh-0.5*dslopet(nh)

C Intercept of deltay and the time derivative of the intercept at time n:

          intertn = (y1*deltay2-y2*deltay1)/dify1

          if (intert(i) .ne. 0.0) then
            dintertnmh = (intertn-intert(i))/dt
          else
            dintertnmh = 0.0
          end if

          dintertn = 1.5*dintertnmh-0.5*dintert(nh)

C Calculate the elastic and viscous components of the tangential force:

          Fte = kte*(slopetn*dify2+intertn*dify1)
          Ftv = ktv*(dslopetn*dify2+dintertn*dify1)

C Force components on block nh and moments on both blocks:
      F1h = dy*(Fne+Fnv)+dx*(Fte+Ftv)
      F2h = -dx*(Fne+Fnv)+dy*(Fte+Ftv)

      Mh = MFne+MFnv+vecmidn(nh,vh2(i))*(Fte+Ftv)-
     &             vecmidt(nh,vh2(i))*(Fne+Fnv)
      Mn = -MFne-MFnv+vecmidn(nn,vn1(i))*(Fte+Ftv)-
     &             vecmidt(nn,vn1(i))*(Fne+Fnv)

C Store the slopes, intercepts, and derivatives:

      slopen(i) = slopenn
      dslopen(i) = dslopennmh
      intern(i) = internn
      dintern(i) = dinternnmh
      slopet(i) = slopetn
      dslopet(i) = dslopetnmh
      intert(i) = intertn
      dintert(i) = dintertnmh

C Cracks grow at a constant velocity:

      if (crackt(i) .gt. 0.0) crackt(i) = crackt(i)+dt*vcrack
      if (crackb(i) .gt. 0.0) crackb(i) = crackb(i)+dt*vcrack

C If the crack crosses the joint, break the joint:

      If (crackt(i) + crackb(i) .gt. 0.9*tlength) then

        Call Overlap(i,nh,nn,nx,ny,Area,xcen,ycen,base)

        status(i) = 'b'
        Ftp(i) = 0.0
        vnormp(i) = 0.0
        vtangp(i) = 0.0
        Areap(i) = Area
        dAreap(i) = 0.0
        ElAreap(i) = 0.0

      End If

      End

******************************************************************

      Subroutine Gridfill

C See Section 2.3.

      include 'blockcom.f'

      Integer*2 i,j,gi,gj

C Set the number in each GRID cell to zero:

      Do i = 1,dimi
        Do j = 1,dimj
          ngrid(i,j) = 0
        End Do
      End Do
```

```
C Sort each polygon by grid location:
C Fill the inverse addresses invx and invy:

      Do i = 1,Nt
        gi = 1+int(x(i)/cell)
        gj = 1+int(y(i)/cell)

        ngrid(gi,gj) = ngrid(gi,gj)+1

        grid(gi,gj,ngrid(gi,gj)) = i
        invx(i) = gi
        invy(i) = gj
      End do

      End
****************************************************************

      Subroutine Initial

C Reads in the system parameters and configuration.

      include 'blockcom.f'

      Integer*2 i,j,jp,k,nh,nn
      Real*8 sino,coso,dx,dy
      Character*1 tstatus

C input Parameters (from standard input):

C cfile0 is the initial configuration file
C cfile(i) is the configuration file dumped following interval i
C rfile(i) is the parameter and result file dumped following interval i
C duration of the simulation
C tpic is the time between calls to the PICTURE subroutine
C tflt is the time between calls to the bouyancy subroutine
C nsearch is the number of time steps between calls to SEARCH
C epsilon is the maximum distance between point/side contacts

      Read (*,*) cfile0
      Read (*,*) nfiles

      Do i = 1,nfiles
        Read (*,*) cfile(i),rfile(i)
      End Do

      Read (*,*) duration
      Read (*,*) tout
      Read (*,*) tpic
      Read (*,*) tflt
      Read (*,*) nsearch
      Read (*,*) epsilon

C Read in the system configuration

      Open(1, File=cfile0, Form='Formatted', Status='Old')

      read (1,*) Nbnd,N
      read (1,*) time0,dt,cell,mu
      read (1,*) kne,knv,kte,ktv
      read (1,*) gsigmat,csigmac
      read (1,*) height,width,waterlevel
      read (1,*) work,penrg,idiss,fdiss,keini

      work = work/dt
      penrg = penrg/dt
      idiss = idiss/dt
      fdiss = fdiss/dt

      Nt = Nbnd+N
      Do i = 1,Nt
        read (1,*) x(i),y(i),th(i),mass(i)
        read (1,*) u(i),v(i),w(i),Jpol(i)

        sino = sin(th(i))
        coso = cos(th(i))
        read (1,*) Nv(i)

        Do j = 1,Nv(i)
          read (1,*) r(i,j),a(i,j)

          rsina(i,j) = r(i,j)*sin(a(i,j))
          rcosa(i,j) = r(i,j)*cos(a(i,j))

C calculate the position of each vertex wrt the center of mass

          rx(i,j) = coso*rcosa(i,j)-sino*rsina(i,j)
          ry(i,j) = sino*rcosa(i,j)+coso*rsina(i,j)
        End Do

        Do j = 1,Nv(i)
          jp = 1+mod(j,Nv(i))

          dx = rx(i,jp)-rx(i,j)
          dy = ry(i,jp)-ry(i,j)

          length(i,j) = sqrt(dx**2+dy**2)

          dx = dx/length(i,j)
          dy = dy/length(i,j)

C vector from center to the midpoint of each side
C (normal and tangential components):

          vecmidn(i,j) = 0.5*(dy*(rx(i,jp)+rx(i,j))-
     &                   dx*(ry(i,jp)+ry(i,j)))

          vecmidt(i,j) = 0.5*(dx*(rx(i,jp)+rx(i,j))+
     &                   dy*(ry(i,jp)+ry(i,j)))

        End Do

        KEini = KEini+0.5*(mass(i)*(u(i)**2+v(i)**2)+jpol(i)*w(i)**2)

      End do

C Read the collisional and glued joint contact parameters:
      read(1,*) count
      Do i = 1,count

        read(1,*) nh,nn,tstatus

        nnear(nh) = nnear(nh)+1
        k = Npr*(nh-1)+nnear(nh)
        near(k) = nn
        status(k) = tstatus

        If (status(k) .eq. 'i') then

C Read the glued joint contact parameters:

          read (1,*) vh1(k),vh2(k),vn1(k),vn2(k)
          read (1,*) slopen(k),dslopen(k),slopet(k),dslopet(k)
          read (1,*) intern(k),dintern(k),intert(k),dintert(k)
          read (1,*) crackt(k),crackb(k)

          Else

C Read the collisional contact parameters:

          read(1,*) Ftp(k),vnormp(k),vtangp(k)
          read(1,*) Areap(k),dAreap(k),ElAreap(k)
          End If
      End Do

      Close(1)

C Define block Np as the water polygon:

      Nv(Np) = 4

      x(Np) = 0.5*width
      y(Np) = 0.5*waterlevel

      rx(Np,1) = 0.5*width
      ry(Np,1) = 0.5*waterlevel
      rx(Np,2) = -0.5*width
      ry(Np,2) = 0.5*waterlevel
      rx(Np,3) = -0.5*width
      ry(Np,3) = -0.5*waterlevel
      rx(Np,4) = 0.5*width
      ry(Np,4) = -0.5*waterlevel

      time = time0
      tfltt = tflt
      scount = nsearch

      End

****************************************************************

      Block Data InitVars

C Initializes the necessary variables

      include 'blockcom.f'
      Data grid,ngrid,count,fcount/dimijk*0,dimij*0,0,1/
      Data near,nnear/NpNpr*0,Np*0/
      Data pt1i,pt1j/NpNpr*1,NpNpr*1/
      Data Ftp,vnormp,vtangp/NpNpr*0.0,NpNpr*0.0,NpNpr*0.0/
      Data Areap,dAreap,ElAreap/NpNpr*0.0,NpNpr*0.0,NpNpr*0.0/
      Data slopen,slopet,intern,intert/NpNpr*0.0,NpNpr*0.0,
     &                 NpNpr*0.0,NpNpr*0.0/
      Data dslopen,dslopet,dintern,dintert/NpNpr*0.0,NpNpr*0.0,
     &                 NpNpr*0.0,NpNpr*0.0/
      Data crackt,crackb/NpNpr*0.0,NpNpr*0.0/
      Data grav,nul,pi,rhow,rhoi/-9.81,0.0,3.14159265359,1010.0,920.0/
      Data toutt,tpict/2*0.0/
      Data F1,F2,M/Np*0.0,Np*0.0,Np*0.0/
      Data F2b,Mb/Np*0.0,Np*0.0/

      End
************************************************************

      Subroutine MoveBlocks

C This subroutine contains the equations of motion for the
C blocks in finite difference form.

      include 'blockcom.f'

      Integer*2 i,j
      Real*8 coso,sino

C Move boundary blocks:

      Do i = 1,Nbnd
        x(i) = x(i)+dt*u(i)
      End Do

C Move active blocks:

      Do i = Nbnd+1,Nt
        u(i) = u(i)+dt*F1(i)/mass(i)
        x(i) = x(i)+dt*u(i)

        v(i) = v(i)+dt*(F2(i)+F2b(i))/mass(i)
        y(i) = y(i)+dt*v(i)

        w(i) = w(i)+dt*(M(i)+Mb(i))/jpol(i)
        th(i) = th(i)+dt*w(i)

C Find the positions of the vertices relative to the center of mass:

        sino = sin(th(i))
        coso = cos(th(i))

        Do j = 1,Nv(i)
          rx(i,j) = coso*rcosa(i,j)-sino*rsina(i,j)
          ry(i,j) = sino*rcosa(i,j)+coso*rsina(i,j)
        End Do
```

```
            F1(i) = 0.0
            F2(i) = 0.0
            M(i) = 0.0
         End Do

C Work performed by ice floes:

         Do i = 1,Nbnd
            work = work+u(i)*F1(i)
            F1(i) = 0.0
         End Do

C Change in potential energy:

         Do i = Nbnd+1,Nt
            penrg = penrg-F2b(i)*v(i)-Mb(i)*w(i)
         End Do

         time = time+dt
         tfltt = tfltt+dt
         toutt = toutt+dt
         tpict = tpict+dt

         End

***********************************************************

         Subroutine Output
         include 'blockcom.f'

         Integer*2 i,j,k
         Real*8 KEout

C Write the system configuration to disk for restart:

         Open(2, File=cfile(fcount), Form='Formatted', Status='New')

         write (2,*) Nbnd,N
         write (2,*) time,dt,cell,mu
         write (2,*) kne,knv,kte,ktv
         write (2,*) gsigmat,csigmac
         write (2,*) height,width,waterlevel
         write (2,*) dt*work,dt*penrg,dt*idiss,dt*fdiss,keini

         Do i = 1,Nt
            write (2,*) x(i),y(i),th(i),mass(i)
            write (2,*) u(i),v(i),w(i),Jpol(i)

            write (2,*) Nv(i)
            Do j = 1,Nv(i)
               write (2,*) r(i,j),a(i,j)
            End do
         End do

         Keout = 0.0
         Do i = Nbnd+1,Nt
            KEout = KEout+0.5*(mass(i)*(u(i)**2+v(i)**2)+jpol(i)*w(i)**2)
         End do
```

```
      count = 0
      Do i = Nbnd+1,Nt
        k = Npr*(i-1)

        Do j = 1,nnear(i)
          k = k+1

          If (status(k) .eq. 'i') then
            count = count+1
          Else
            if (Areap(k) .ne. 0.0) count = count+1
          End If

        End Do
      End Do

       write (2,*) count
       Do i = Nbnd+1,Nt
         k = Npr*(i-1)

         Do j = 1,nnear(i)
           k = k+1

           If (status(k) .eq. 'i') then

C Save the glued joint contact parameters:

             write (2,*) i,near(k),status(k)
             write (2,*) vh1(k),vh2(k),vn1(k),vn2(k)
             write (2,*) slopen(k),dslopen(k),slopet(k),dslopet(k)
             write (2,*) intern(k),dintern(k),intert(k),dintert(k)
             write (2,*) crackt(k),crackb(k)

           Else

             if (Areap(k) .ne. 0.0) then

C Save the Active Collisional-contact parameters:

               write (2,*) i,near(k),status(k)
               write (2,*) Ftp(k),vnormp(k),vtangp(k)
               write (2,*) Areap(k),dAreap(k),ElAreap(k)

             end if

           End If
         End Do
      End Do

      Close(2)

C Save the system parameters and experimental results:

      Open(3, File=rfile(fcount), Form='Formatted', Status='New')

      write (3,'(a)') 'Pressure Ridge Simulation'
      write (3,'(a)') ''
      write (3,'(a)') ' General parameters: '
      write (3,'(a,f13.5)') 'starting time ',time-tout
      write (3,'(a,f13.5)') 'time duration ',tout
      write (3,'(a,f13.5)') 'time step ',dt
      write (3,'(a)') ''
      write (3,'(a)') ' Material parameters: '
      write (3,'(a,f13.5)') 'floe velocity ',u(Nbnd)
      write (3,'(a,f13.5)') 'lead ice thickness ',tlead
      write (3,'(a,f13.5)') 'average ice density ',rhoi
      write (3,'(a,f13.5)') 'water density ',rhow
      write (3,'(a,f13.2)') 'block stiffness (n) ',kne
      write (3,'(a,f13.2)') 'block viscosity (n) ',knv
      write (3,'(a,f13.2)') 'block stiffness (t) ',kte
      write (3,'(a,f13.2)') 'block viscosity (t) ',ktv
      write (3,'(a,f13.0)') 'tens. strength (glue) ',gsigmat
      write (3,'(a,f13.0)') 'comp. strength (coll) ',csigmac
      write (3,'(a,f13.2)') 'friction block/block ',mu
      write (3,'(a)') ''
      write (3,'(a)') ' Energetics: '
      write (3,'(a,f13.2)') 'Ridging work ',dt*work
      write (3,'(a,f13.2)') 'Frictional losses ',dt*fdiss
      write (3,'(a,f13.2)') 'Inelastic losses ',dt*idiss
      write (3,'(a,f13.2)') 'Potential E change ',dt*penrg
      write (3,'(a,f13.2)') 'Change in Kin Energy ',KEout-KEini

      Close(3)

      fcount = fcount+1
      toutt = 0.0

      End

******************************************************************

      Subroutine Overlap(ind,i,j,nx,ny,area,xcen,ycen,base)

C The Overlap subroutine calculates the overlapping area between
C a pair of polygons.  It locates the centroid ot the area of
C overlap and sets up the local coordinate frame with normal
C to the plane of contact whose breadth is the base.


      include 'blockcom.f'

      Integer*2 ind,i,ii,ip,j,jj,jp,k,npt,nint,tpt1i,tpt1j,nadv
      Real*8 dx,dy,vijx,vijy,viix,viiy,vjjx,vjjy
      Real*8 xprodij,xprodji,xprodijo,xprodjio,xprodijp
      Real*8 base,mag,nx,ny,xcen,ycen,Area,tArea
      Real*8 xint(Nvmax),yint(Nvmax),xpt(Nvmax),ypt(Nvmax)
      Character*1 lastmove,lastmovei,lastmovej
      Logical done

      nadv = 0

      nint = 0
      npt = 0

      area = 0.0
      done = .false.

      tpt1i = 0
      tpt1j = 0
```

```
          dx = x(j)-x(i)
          dy = y(j)-y(i)

          ii = pt1i(ind)
          jj = pt1j(ind)

          ip = 1+mod(ii, Nv(i))
          jp = 1+mod(jj, Nv(j))

          viix = rx(i,ip)-rx(i,ii)
          viiy = ry(i,ip)-ry(i,ii)

          vjjx = rx(j,jp)-rx(j,jj)
          vjjy = ry(j,jp)-ry(j,jj)

          vijx = dx+rx(j,jj)-rx(i,ii)
          vijy = dy+ry(j,jj)-ry(i,ii)

          xprodij = viix*(vijy+vjjy)-viiy*(vijx+vjjx)
          xprodji = vjjx*(-vijy+viiy)-vjjy*(-vijx+viix)

          xprodijo = viix*vijy-viiy*vijx
          xprodjio = -vjjx*vijy+vjjy*vijx

          lastmovei = 'o'
          lastmovej = 'o'

          Do While (.not. done)

            nadv = nadv+1

C Check for intersection:

            If (xprodij*xprodijo .lt. 0.0 .and.
     &                  xprodji*xprodjio .lt. 0.0) then

              If (ii .ne. tpt1i .or. jj .ne. tpt1j) then
                nint = nint+1
                npt = npt+1

                mag = abs((vijx*viiy-vijy*viix)/(vjjx*viiy-vjjy*viix))

                xpt(npt) = x(j)+rx(j,jj)+mag*vjjx
                ypt(npt) = y(j)+ry(j,jj)+mag*vjjy

                xint(nint) = xpt(npt)
                yint(nint) = ypt(npt)

C pt1i and pt1j prevent the same intersection being saved twice:

                if (nint .eq. 1) then
                  tpt1i = ii
                  tpt1j = jj

                  pt1i(ind) = ii
                  pt1j(ind) = jj
                end if
            if (xprodji .gt. 0.0) then
              lastmovei = 'i'
              lastmovej = 'o'
            else
              lastmovei = 'o'
              lastmovej = 'i'
            end if

          Else

            Done = .true.

          End If
        End If

        If (xprodij*xprodji .lt. 0.0) then

C One is in and one is out:

          If (xprodij .gt. 0.0) then

C j is in, advance on i:

            ii = ip
            lastmove = 'i'

          Else

C i is in, advance on j:

            jj = jp
            lastmove = 'j'

          End If

        Else

C Extend vectors Vii and Vjj, make new cross-products:

          mag = 1.0e30

          xprodijp = viix*(vijy+mag*vjjy)-viiy*(vijx+mag*vjjx)

C Both i and j are in:

          If (xprodij .gt. 0.0) then

            If (xprodijp .gt. 0.0) then

C extended i is out, advance on i:

              ii = ip

C Point is a vertex of the overlap polygon:

              If (lastmovei .eq. 'i') then

                npt = npt+1
```

```
                  xpt(npt) = x(i)+rx(i,ii)
                  ypt(npt) = y(i)+ry(i,ii)

                End If

                lastmove = 'i'

              Else

C extended j is out, advance on j:

                jj = jp

C Point is a vertex of the overlap polygon:

                If (lastmovej .eq. 'i') then

                  npt = npt+1

                  xpt(npt) = x(j)+rx(j,jj)
                  ypt(npt) = y(j)+ry(j,jj)

                End If

                lastmove = 'j'

              End If

            Else

C Both i and j are out:

              If (xprodijp .gt. 0.0) then

C extended j is in, advance on j:

                jj = jp
                lastmove = 'j'

              Else

C extended i is in, advance on i:

                ii = ip
                lastmove = 'i'

              End If
            End If
          End If

C After the algorithm has made 2*(Nv(i)+Nv(j)) advances then quit:

          If (nadv.ge.2*

     (Nv(i)+Nv(j))) then
            done = .true.
          Else

C Make vectors and cross-products:
```

```
              vijx = dx+rx(j,jj)-rx(i,ii)
              vijy = dy+ry(j,jj)-ry(i,ii)

              If (lastmove .eq. 'i') then

                ip = 1+mod(ii, Nv(i))

                viix = rx(i,ip)-rx(i,ii)
                viiy = ry(i,ip)-ry(i,ii)

                xprodijo = viix*vijy-viiy*vijx
                xprodjio = xprodji

              End If

              If (lastmove .eq. 'j') then

                jp = 1+mod(jj, Nv(j))

                vjjx = rx(j,jp)-rx(j,jj)
                vjjy = ry(j,jp)-ry(j,jj)

                xprodijo = xprodij
                xprodjio = -vjjx*vijy+vjjy*vijx

              End If

              xprodij = viix*(vijy+vjjy)-viiy*(vijx+vjjx)
              xprodji = vjjx*(-vijy+viiy)-vjjy*(-vijx+viix)

            End If

          End Do

          If (npt .gt. 0) then

C Calculate area of overlap and location of centroid of area:

            xcen = 0.0
            ycen = 0.0

            Do k = 1,npt-2

              tArea = (xpt(k+1)-xpt(1))*(ypt(k+2)-ypt(1))-
     &                (ypt(k+1)-ypt(1))*(xpt(k+2)-xpt(1))

              xcen = xcen+tArea*(xpt(k+1)-xpt(1)+xpt(k+2)-xpt(1))
              ycen = ycen+tArea*(ypt(k+1)-ypt(1)+ypt(k+2)-ypt(1))

              Area = Area+tArea

            End Do

            xcen = xpt(1)+xcen/Area/3
            ycen = ypt(1)+ycen/Area/3

            Area = 0.5*Area

C Define the line of contact:
```

```
          If (nint .eq. 2) then

            dx = xint(2)-xint(1)
            dy = yint(2)-yint(1)

            xprodij = dx*(y(i)-yint(1))-dy*(x(i)-xint(1))

          End If

C Exceptional case where corners overlap, but no vertex lies within
C either block. Choose the line for which the centers lie on opposite
C sides.  More sophisticated strategies may be necessary.

          If (nint .eq. 4) then

            dx = xint(3)-x
            dy = yint(3)-y [illegible])

            xprodij = dx*(y(i)-yint(1))-dy*(x(i)-xint(1))
            xprodji = dx*(y(j)-yint(1))-dy*(x(j)-xint(1))

            if (xprodij*xprodji .gt. 0.0) then

              dx = xint(4)-xint(2)
              dy = yint(4)-yint(2)

              xprodij = dx*(y(i)-yint(2))-dy*(x(i)-xint(2))

            end if

          End If

C local coord frame defined by normal vector
C inwardly perpendicular to the HOME (i) polygon:

          base = sqrt(dx*dx+dy*dy)

          if (xprodij .lt. 0.0) then
            nx = dy/base
            ny = -dx/base
          else
            nx = -dy/base
            ny = dx/base
          end if

        End If

        End
****************************************************************

        Subroutine Picture

C A rudimentary routine to make a line drawing of each polygon
C in the system at the current time.  It uses UNIX graphics primitives.

        include 'blockcom.f'
```

```
   Integer*2 i,j,jp
   Integer*4 xp(Nvmax),yp(Nvmax),xc,yc,xr,yr,range,ywl
   Character*40 text

   write(text,'(a,f12.4)') 'Time: ',time
   text(40:40) = char(00)

   xc = 100
   yc = 100
   xr = 3800
   yr = 2500

   range = int(min(xr/width,yr/height))

   Call openpl
   Call erase
   Call move(xc,2900)
   Call Label(text)

   ywl = yc+int(range*waterlevel)
   Call line(xc,ywl,xc+xr,ywl)

   do i = 1,Nt

     do j = 1,Nv(i)
       xp(j) = xc+int(range*(x(i)+rx(i,j)));
       yp(j) = yc+int(range*(y(i)+ry(i,j)))
     end do

     do j = 1,Nv(i)
       jp = 1+mod(j,Nv(i))

       Call Line(xp(j),yp(j),xp(jp),yp(jp))
     end do

   end do

   Call move(1,1)

   Call Closepl
   tpict = 0.0

   End
*****************************************************************

        Subroutine Search

C This subroutine performs the Global Search
C The neighborhood of each polygon is searched for neighbors
C The Contact subroutine is called to examine a pair (nh,nn)
C for proximate contact. See Section 2.3.

C The near-neighbor list containing the indices of the Home and
C Near polygons, the collisional force and the glue force variables.
C These variables must be passed through the search procedure and
C reconnected with the correct pair of polygons if that pair of
C polygons is still found to be in contact. As the contacts are
C discovered the variables are matched up with the correct pair
C of polygons and the interaction list is assembled.
```

```
C Npr spaces are allotted in the near-neighbor list per block.
C Linked lists would be more efficient, but less straightforward.

C And memory being less of a constraint than CPU speed, it isn't
C worth the trouble.

      include 'blockcom.f'

      Real*8 t1(Npr),t2(Npr),t3(Npr),t4(Npr),t5(Npr),t6(Npr)
     &           t7(Npr),t8(Npr),t9(Npr),t10(Npr)
      Integer*2 tnear(0:Npr),tpt1i(Npr),tpt1j(Npr),t11(Npr),t12(Npr)
     &           t13(Npr),t14(Npr)
      Integer*2 i,j,k,kk,tnnear,ind,nn,nh
      Character*1 tstatus(0:Npr)
      Logical neighbor

      tnear(0) = 0
      tstatus(0) = 'n'

      Do nh = Nbnd+1,Nt

        ind = Npr*(nh-1)

C Store Near, and contact variables in temporary arrays

        tnnear = nnear(nh)

        Do i = 1, tnnear
          ind = ind+1

          tnear(i) = near(ind)
          tstatus(i) = status(ind)

          If (status(ind) .eq. 'b') then

            tpt1i(i) = pt1i(ind)
            tpt1j(i) = pt1j(ind)

            t1(i) = Ftp(ind)
            t2(i) = vnormp(ind)
            t3(i) = vtangp(ind)
            t4(i) = Areap(ind)
            t5(i) = dAreap(ind)
            t6(i) = ElAreap(ind)

          Else

            t1(i) = slopen(ind)
            t2(i) = dslopen(ind)
            t3(i) = slopet(ind)
            t4(i) = dslopet(ind)
            t5(i) = intern(ind)
            t6(i) = dintern(ind)
            t7(i) = intert(ind)
            t8(i) = dintert(ind)
            t9(i) = crackt(ind)
            t10(i) = crackb(ind)
            t11(i) = vh1(ind)
            t12(i) = vh2(ind)
          t13(i) = vn1(ind)
          t14(i) = vn2(ind)

        End If
      End Do

C Search the neighborhood about each polygon:

      nnear(nh) = 0
      ind = Npr*(nh-1)

      Do i = max(1,invx(nh)-1), invx(nh)+1
        Do j = max(1,invy(nh)-1), invy(nh)+1

          Do k = 1,ngrid(i,j)

            nn = grid(i,j,k)

            kk = tnnear
            Do While (kk .gt. 0 .and. nn .ne. tnear(kk))
              kk = kk-1
            End Do

            If (tstatus(kk) .eq. 'i') then

              ind = ind+1

C Reattach the stored glued joint variables to the correct pair:

              near(ind) = nn
              status(ind) = tstatus(kk)

              nnear(nh) = nnear(nh)+1

              sloper(ind) = t1(kk)
              dslopen(ind) = t2(kk)
              slopet(ind) = t3(kk)
              dslopet(ind) = t4(kk)
              intern(ind) = t5(kk)
              dintern(ind) = t6(kk)
              intert(ind) = t7(kk)
              dintert(ind) = t8(kk)
              crackt(ind) = t9(kk)
              crackb(ind) = t10(kk)
              vh1(ind) = t11(kk)
              vh2(ind) = t12(kk)
              vn1(ind) = t13(kk)
              vn2(ind) = t14(kk)

            Else

              If (nn .lt. nh) then

                neighbor = .false.
                Call Contact(nh,nn,neighbor)

                If (neighbor) then

                  ind = ind+1
```

```
                    near(ind) = nn
                    status(ind) = 'b'

                    nnear(nh) = nnear(nh)+1

C Reattach the stored collision variables to the correct pair:

                    If (tstatus(kk) .eq. 'n') then

                       pt1i(ind) = 1
                       pt1j(ind) = 1

                       Ftp(ind) = 0.0
                       vnormp(ind) = 0.0
                       vtangp(ind) = 0.0
                       Areap(ind) = 0.0
                       dAreap(ind) = 0.0
                       ElAreap(ind) = 0.0

                    Else

                       pt1i(ind) = tpt1i(kk)
                       pt1j(ind) = tpt1j(kk)

                       Ftp(ind) = t1(kk)
                       vnormp(ind) = t2(kk)
                       vtangp(ind) = t3(kk)
                       Areap(ind) = t4(kk)
                       dAreap(ind) = t5(kk)
                       ElAreap(ind) = t6(kk)

                  End If
                End If
              End If
            End If
          End Do
        End Do
      End Do
    End Do

    End
```

APPENDIX B: SUPPLEMENTARY SUBROUTINE FOR CALCULATION OF THE MASS AND THE POLAR MOMENT OF INERTIA OF A CONVEX POLYGON

```
*******************************************************************************

      Subroutine Mass_and_Moment(i)

      include 'blockcom.f'

      Integer*2 i,k,kp
      Real*8 Area,sArea,xcm,ycm,ssx,ssy,mag,b1,b2,h

C Find the mass and the location of the center of mass of block i
C Given vectors rx and ry from an arbitrary interior point x(i),y(i)
C to the vertices.  See section 2.8 of this report.

      Area = 0.0
      xcm = 0.0
      ycm = 0.0

      Do k = 1,Nv(i)
        kp = 1+mod(k,Nv(i))

        sArea = 0.5*abs(rx(i,k)*ry(i,kp)-ry(i,k)*rx(i,kp))
        xcm = xcm+sArea*(rx(i,k)+rx(i,kp))/3.0
        ycm = ycm+sArea*(ry(i,k)+ry(i,kp))/3.0
        Area = Area+sArea
      End Do

      xcm = xcm/Area
      ycm = ycm/Area

C The location of the center of mass of block i:

      x(i) = x(i)+xcm
      y(i) = y(i)+ycm
      mass(i) = rho*Area

C Vectors from the center of mass to the vertices of block i:
C Angles and Radii define the block's shape:

      Do k = 1,Nv(i)
        rx(i,k) = rx(i,k)-xcm
        ry(i,k) = ry(i,k)-ycm

        r(i,k) = sqrt(rx(i,k)**2+ry(i,k)**2)

        a(i,k) = atan2(ry(i,k),rx(i,k))
        if (a(i,k) .lt. 0.0) a(i,k) = [illegible] pi+a(i,k)
      End Do

C Find the polar moment of inertia of block i:

      Jpol(i) = 0.0
      Do k = 1,Nv(i)
        kp = 1+mod(k,Nv(i))
```

```
   ssx = rx(i,kp)-rx(i,k)
   ssy = ry(i,kp)-ry(i,k)
   mag = sqrt(ssx*ssx+ssy*ssy)
   ssx = ssx/mag
   ssy = ssy/mag

   b1 = abs(ssx*rx(i,k)+ssy*ry(i,k))
   b2 = abs(ssx*rx(i,kp)+ssy*ry(i,kp))
   h = abs(ssx*ry(i,k)-ssy*rx(i,k))

   Jpol(i) = Jpol(i)+h*(b1*(b1*b1+3.0*h*h)+b2*(b2*b2+3.0*h*h))/12.0
End Do

End
```

APPENDIX C: FLATLAND REVISITED

The following entertaining description of the problems of polygonal entities in a two-dimensional world was written by Otis Walton as an addendum to his work describing the Discrete Interacting Block System code (DIBS) (Walton 1980). It is reprinted with his permission.

In the spirit of Abbott's 19th century mythical world of Flatland, the two-dimensional world of DIBS is inhabited by beings that cannot see very far and can only recognize their neighbors by "feeling" them. The highest social classes in Flatland are the regular *n*-gons, with a large number of sides. The high priests actually claim to be circles. The lowest social class is composed of isosceles triangles with very small acute angles. The size of a Flatlander's brain is easily estimated from the size of the angle of his vertex.

Line segments (Flatland women) and triangles with small acute angles are very dangerous in Flatland because, when coming head on, they are hard to see and can puncture a more normal Flatlander with a mere casual bump.

Most of the random polygons allowed in DIBS would be considered to be the most grotesque of monsters in Flatland because of their ugly irregularity. Consequently, most would probably be eliminated at the earliest opportunity in favor of regular polygons of high order.

The irregular Flatland particles in DIBS show even more male chauvinism than their regular counterparts in Flatland. Women (mere line segments) are completely banned in DIBS, as are all small children. Line segments, being completely massless, flit about from place to place at incredible speed—in order to be seen at all, the clocks of the entire society have to be made to run slower than normal (the time step must be decreased); so much slower that the entire society appears to come to a standstill, except for these very light beings flitting about. Small children, with their low mass, have much the same effect as almost massless women. If the society tries to exist at normal clock speed when there are some of these little tykes about, we soon find that they have bumped into someone and are unstably being accelerated to incredible speeds—sometimes traveling so fast that they pass most of the way through another body in just one tick of the society's master clock.

For these reasons, the population in this new irregular Flatland is restricted so that from the smallest to the largest being is no more that 2 orders of magnitude change in size (mass). Preferably, all beings will be even closer to the same size than that. A few unusually large beings are OK, but never should the population include only a few very small beings in a situation where most of the others are large.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE December 1992	3. REPORT TYPE AND DATES COVERED

4. TITLE AND SUBTITLE Numerical Simulation of Systems of Multitudinous Polygonal Blocks	5. FUNDING NUMBERS N00014-86-K-K069
6. AUTHORS Mark A. Hopkins	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) U.S. Army Cold Regions Research and Engineering Laboratory 72 Lyme Road Hanover, N.H. 03755-1290	8. PERFORMING ORGANIZATION REPORT NUMBER CRREL Report 92-22
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Office of Naval Research 800 N. Quincy St. Arlington, VA 22217-5000	10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited. Available from NTIS, Springfield, Virginia 22161.	12b. DISTRIBUTION CODE

13. ABSTRACT (*Maximum 200 words*)

14. SUBJECT TERMS Discrete element method; Ice ridging; Pressure ridging; Ice mechanics; Particle simulation	15. NUMBER OF PAGES 74
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. Z39-18
298-102